NETWORK MONITORING USING NAGIOS AND AUTOCONFIGURATION FOR CYBER DEFENSE COMPETITIONS

Jaipaul Vasireddy
B.Tech, A.I.E.T, Jawaharlal Nehru Technological University, India, 2006

PROJECT

Submitted in partial satisfaction of
the requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

at

CALIFORNIA STATE UNIVERSITY, SACRAMENTO

FALL
2009

NETWORK MONITORING USING NAGIOS AND AUTOCONFIGURATION FOR CYBER DEFENSE COMPETITIONS

A Project

by

Jaipaul Vasireddy

Approved by:

_____, Committee Chair
Dr. Isaac Ghansah

_____, Second Reader
Prof. Richard Smith

_____
Date

Student: Jaipaul Vasireddy

I certify that this student has met the requirements for format contained in the University format manual, and that this Project is suitable for shelving in the Library and credit is to be awarded for the Project.

_____, Graduate Coordinator _____

Dr. Cui Zhang Date

Department of Computer Science

Abstract

of

NETWORK MONITORING USING NAGIOS AND AUTOCONFIGURATION FOR CYBER DEFENSE COMPETITIONS

by

Jaipaul Vasireddy

The goal of the project is to monitor the services running on the CCDC (College Cyber Defense Competition) network, using Nagios which uses plugins to monitor the services running on a network. Nagios is configured by building configuration files for each machine which is usually done to monitor small number of systems. The configuration of Nagios can also be automated by using shell scripting which is generally done in an industry, where the numbers of systems to be monitored are large. Both the above methods of configuration have been implemented in this project.

The project has been successfully used to know the status of each service running on the defending team's network. The project also includes network traffic reduction using failover monitoring where one Nagios machine monitors another Nagios machine so that if one machine fails the other machine does the work of the broken system.

_____, Committee Chair
Dr. Isaac Ghansah

_____
Date

ACKNOWLEDGEMENTS

I consider this as a wonderful opportunity to thank Dr.Isaac Ghansah and Professor Smith who gave me this opportunity to work under their esteemed guidance. I also thank all the members in white team for their valuable suggestions which helped a lot for the successful completion of this project.

TABLE OF CONTENTS

Page

LIST OF FIGURES

Page

Chapter 1

INTRODUCTION

An organization's network infrastructure consists of routers, switches, servers and other equipment that provide services to their clients. Each day results in new security threats, including DOS (denial of service) attacks, worm and virus outbreaks deliberately created to directly or indirectly disrupt the services that our network infrastructure attempts to provide [1]. Therefore there is a necessity to understand how to protect our infrastructure by using security tools and best practices to protect each system in our network. The computer science classes csc254, csc154 (Network Security and Cryptography, Computer System Attacks and Counter Measures) educates students with theoretical knowledge necessary to protect the network infrastructure, students can apply the security practices learned in class on the CCDC network, depicted in Figure1.1 taken from[2]. And in order to understand how a hacker thinks there should be someone who needs to attack the CCDC network, so the entire class is divided into two groups where one group attacks the CCDC network called the Red team (attackers) and other group defends the network called the Blue team (defenders). But there should be a way where the performance of the teams can be known, this project provides a way where performance of the teams can be known.

The project is to build a monitoring system using Nagios for the college cyber defense competition where there are two teams which are called Red team (attackers),

and Blue team (defenders). Each team is assigned a group of systems on which they have been given administrative privileges, and they are placed in separate rooms so that each team has privacy.

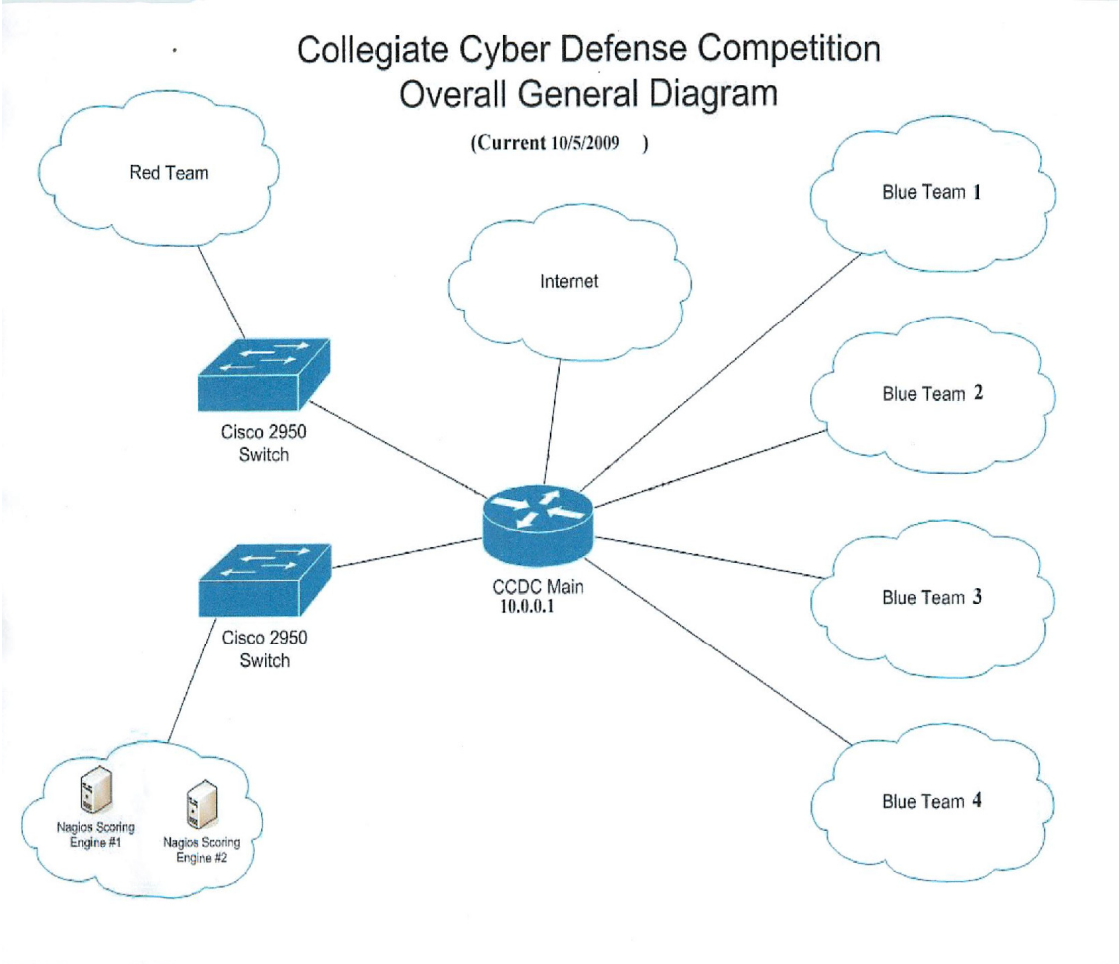


Figure 1.1: CCDC Network Design [2]

The goal of the attackers is to bring down the services of the defenders, and the job of the defending team is to defend themselves from being hacked and always keep their services running. The monitoring tool Nagios is used to check the services running on the blue teams network (Figure 1.2 taken from [2]) periodically, and store the collected data in MySQL database which is further utilized for scoring purposes to determine the performance of each team [3].

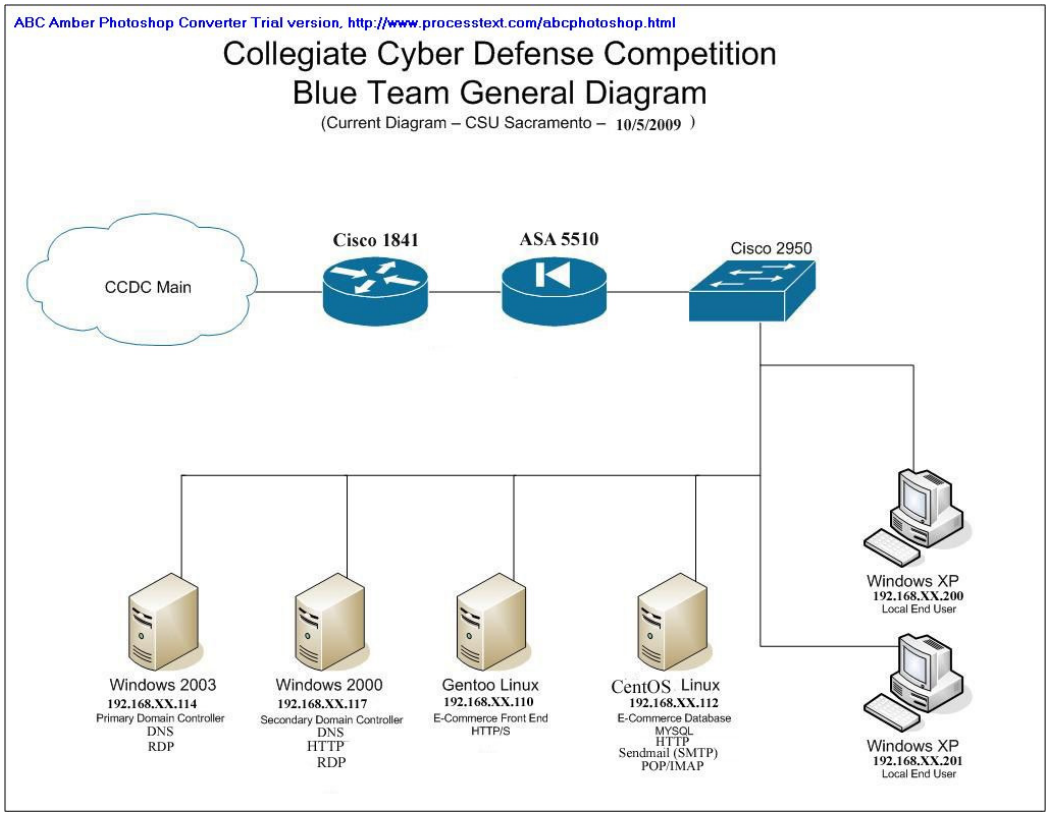


Figure1.2: Blue Team Network [2].

Apart from the blue team and red team there is a third team called the white team who are the judges of the competition. The job of the white team is to set up the network for the competition and to frame rules for the competition before the competition begins so that the competition is played fair. They also create a real time industry environment so that students would have better exposure. They do this by sending injects, which are something like a work order which are further discussed in the report.

There are eight regional CCDC competitions which are held in United States, the Western Regional Collegiate Cyber Defense Competition held at Cal Poly (California State Polytechnic University, Pomona) is one such regional competition. Sacramento State University sends a team, who are cognizant in network security to Cal Poly every year.

The winners of each regional competition compete in National Collegiate Cyber Defense Competition held at University of Texas, San Antonio, where scoring is done on a positive scale, and the team with the highest number of points wins the competition unlike ours where scoring is done on a negative scale and defending teams start at zero points and the team which has lower points wins the competition. The other major difference is performance can be known only at the end of the day because the scoring is done by maintaining logs for each service being monitored [4]. Unlike ours, where we can know the performance of the team, any time during the competition.

The CCDC is similar to capture the flag contest which is held at Defcon the hacker's conference at Las Vegas, Nevada. In capture the flag contest, the participating teams have a dedicated flag for each service and they need to keep their services running while attempting to change the flags of competitor's services to their own. Their scoring system is designed in such a way that it periodically checks the functionality of each of the competitor's network. If the service is functional, and the flag is that of the owning team, the team receives points. If the service is unavailable the team receives no points. If the service is running but the flag is of another team, the other team gets the points [5].

The report is further organized as follows.

Chapter2: Nagios; The functionality and directory structure of Nagios is discussed.

Chapter3: Active monitoring; In Active monitoring Nagios directly checks the status of each service.

Chapter4: Passive monitoring; In Passive Monitoring Nagios uses plug-ins to run remotely which report their results to Nagios.

Chapter5: Distributed monitoring; In Distributed monitoring the network infrastructure is monitored by different monitoring machines.

Chapter6: Failover monitoring; In Failover monitoring one Nagios machine monitors another Nagios machine, the sequence of steps which needs to be followed are discussed here.

Chapter7: Configuration of Nagios; Here the sequence of steps to be followed to configure Nagios are discussed.

Chapter8: Autoconfiguration; Here the purpose of Autoconfiguration, the format of text files which needs to be written for proper functionality of Autoconfiguration scripts is discussed.

Chapter9: Inject automation; a shell script which is used to send injects for the defending team is explained.

Chapter 10: Conclusion; Here the strengths and weaknesses of the project are discussed.

Chapter 2

NAGIOS

Nagios is a monitoring tool that monitors current status of hosts, printers, switches and services that run on a network. The CCDC network is monitored using Nagios. Nagios compilation requires gcc, make, autoconf, and automake utilities. The required libraries for Nagios compilation are libgd and openssl [6]. Nagios plugins are used to monitor services running on each host on a specified port with a specified IP. Nagios allows notifications to be sent through email when a particular service or host is down to the network administrator.

Nagios can monitor the publicly available services in two ways active monitoring and passive monitoring. In active monitoring Nagios directly monitors the services of the host, where as in passive monitoring an add-on running on the remote host reports its results to Nagios. Using passive monitoring you can also monitor private information of the remote machine such as number of processes which otherwise cannot be accessed using active monitoring.

Nagios can be scheduled to monitor in a 24*7 environment or it can be scheduled to monitor in specific time interval like nine to five. It can also be scheduled to monitor in a 24*7 environment except on holidays.

The entire configuration of Nagios depends on three configuration files nagios.cfg, cgi.cfg and resource.cfg. Nagios considers each network component to be monitored as an object, and all object definitions end up in the subdirectory called objects [1].

2.1 Directory Structure of Nagios

The directory structure of nagios is depicted in Figure 2.1

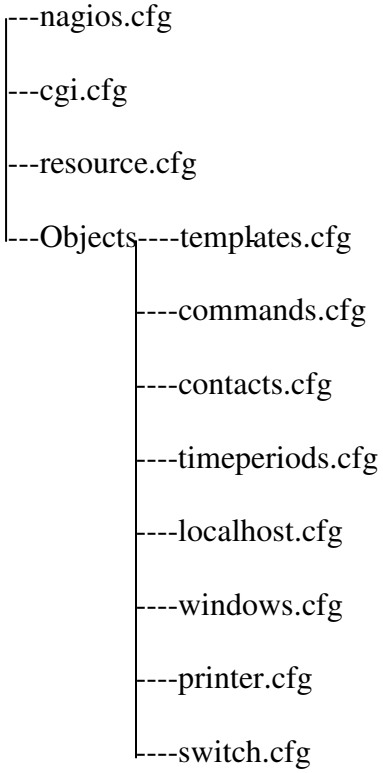


Figure 2.1 Nagios Directory Structure

A Few of the Significant Configuration Options of nagios.cfg are explained below.

Accept_passive_host_checks: If agents like Nrpe or Nsclient are to be disabled this option needs to be zero by default it is set as one.

Admin_email: The email address of the Nagios administrator specified in macro $ADM INEMAIL$, all the notifications will be send to this particular email address. This was specified as csus.nagios@gmail.com.

Log_file: All errors and problems are recorded in this file.

Log_passive_checks: Specifies whether Nagios should log passive checks in the log file (1=enable, 0=disable).

Enable_notifications: Specifies whether Nagios can send notifications (1=enable, 0=disable). It was set as 1 so that all the notifications will be sent to csus.nagios@gmail.com.

Enable_environment_macros: Specifies macros like $HOSTADDRESS$ can be accessed as environment variables (1=enable, 0=disable). It was set as 1.

Log_rotation_method: Specifies the time after which the log file is considered stale and a new file is created the different options are n for don't rotate the log, h for hourly log rotation ,d for daily rotation ,w for weekly rotation for monthly rotation. We specified the log rotation to be none.

The significant configuration options specified in templates.cfg are discussed below.

Check_period: The time during which the service check can be done, was specified as 24*7 since the competition is run in a 24*7 environment, the defending teams services needs to be checked all throughout the day.

Max_check_attempts: Nagios does not report a state until a certain number of attempts specified by the user, it was configured as three so that it needs to check thrice before reporting a state.

Retry_check_interval: It specifies the time gap between two attempts to determine a hard state, it was specified as one.

Normal_check_interval: Specifies the time gap between successive checks during normal conditions, i.e. if nagios determines a hard state in the first attempt the next attempt will be after 5 minutes.

2.2 Scheduling Nagios

Although the competition is run in a 24*7 environment there is always a possibility of having a holiday during the competition. In such cases instead of manually stopping and starting nagios, we can directly specify the time during which nagios needs to monitor, in a file called timeperiods.cfg .Here we define a time period and make sure that the templates.cfg file uses the defined time period. The configuration option in templates.cfg is check_period, which in turn is used by all object configuration files. For example if Nagios needs to be running in a 24*7 environment except on memorial day, we create a holiday time template which in turn is used by CCDC-working timeperiod, so that when we use the CCDC-working time period in the template definition, the services will not be checked on the specified days.

```
Define timeperiod {
                    Timeperiod_name ccdc-holidays
                    Monday -1 may     00:00-00:00
                }
```

And we defined a second template called ccdc-working

```
Define timeperiod {
                    Timeperiod_name ccdc-working
                    Use     ccdc-holidays
                    Sunday 00:00-24:00
                    Monday 00:00-24:00
                    Tuesday 00:00-24:00
                    Wednesday 00:00-24:00
                    Thursday 00:00-24:00
                    Friday00:00-24:00
                    Saturday00:00-24:00
                }
```

Apart from network monitoring, Nagios can be used to analyze numerical data like temperature, which is retrieved from different sensors and compares it with the preconfigured temperature, and if needed, sends an alert to the system administrator.

Chapter 3

ACTIVE MONITORING

Active monitoring is a way in which Nagios directly monitors the services of each machine by using plugins. Nagios uses check logic, which in turn uses plugins when it needs to monitor a host or service. A plugin can be shell script or Perl script (Nagios has an embedded Perl interpreter) that runs from the command line to perform a host or service check and then returns their results to the Nagios daemon. This further performs the required action like sending notifications.

The frequency at which these checks should be done is specified in check_interval and retry_interval of the host and service definitions.

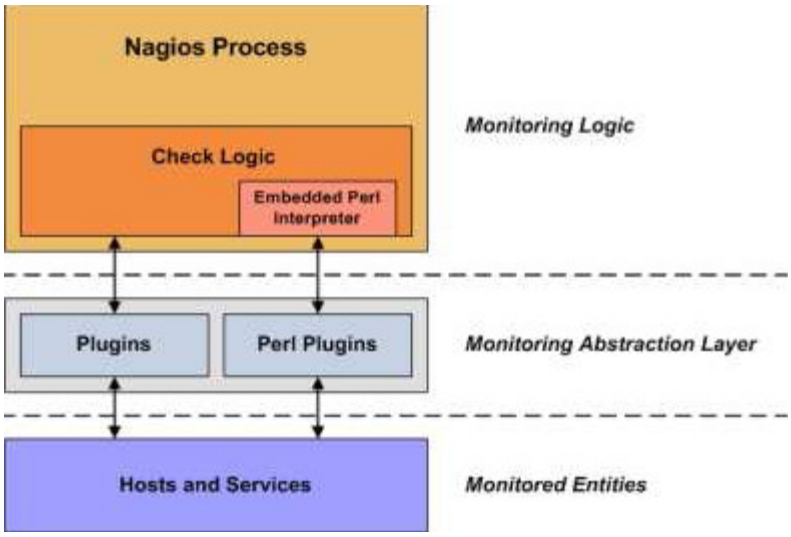


Figure 3.1 Nagios Active Monitoring Overview (source:www.Nagios.org)

If the plugin finds a hard state like ok or critical, the next scheduled check will be the check_interval. If the service is flapping, that is, if it's not sure if the service is ok or critical, and it's still under the maximum allowed check_attempts, the next scheduled check is the retry_interval.

Example of active check: check_tcp –H 192.168.129.17 -w 5 –c 10

This means that the machine 192.168.129.17 needs to complete the handshake within 5 seconds, otherwise it's going to report a warning and if it does not respond within 10 seconds it's going to report a critical stage.

Chapter 4

PASSIVE MONITORING

A logical question could be: when there is active monitoring what's the need for passive monitoring? With active monitoring one can just check the services running on the machine, but you cannot retrieve the private information of the machine, such as number of users on the machine, the load on the machine, the total number of processes on the machine. This type of data is required in an IT industry in order to make sure that systems meet their performance criteria. If not, the situation needs to be reported to the authorized individual so that he could take the necessary steps.

This crucial data cannot be acquired by a simple hand shake protocol such as TCP/IP. So in order to acquire this private data, you need to have daemon running on the client side which can be installed only when you have administrative privileges on the system. Systems that need to be monitored can either be a windows system or a Linux system; depending on the operating system of the client you can either install NRPE agent if it's a Linux system or NSClient++ agent if it's a windows machine. And we use respective plugins check_nrpe, check_nsclient++ to retrieve the data from the agents installed remotely. In short, if you want to monitor the services running behind the firewall we use passive monitoring.

The relationship between the agent, plugin, monitoring server and the monitored server in Linux environment is depicted in Figure 4.1.

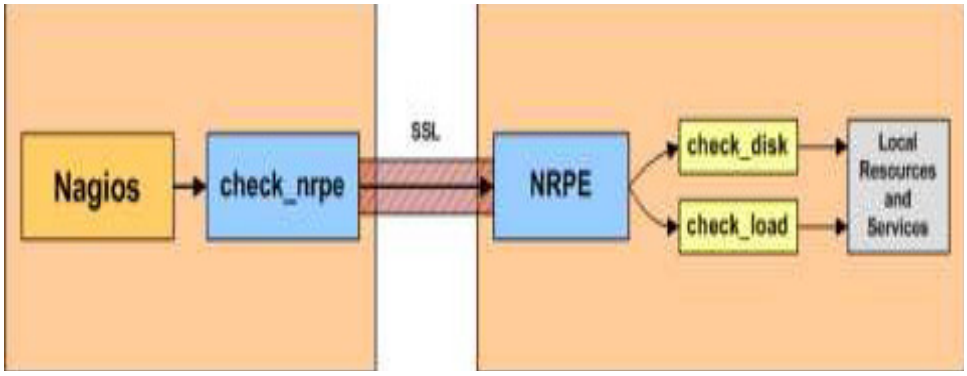


Figure 4.1 Overview of Nagios Passive Monitoring Using NRPE

(source:www.Nagios.org)

4.1 NRPE Implementation Steps [4]:
Step1: Install Nrpe agent on the remote server (instructions how to install are specified in the user manual).

Step2:Check_nrpe by default comes with Nagios but we still need to define in command definition.

So add the following definition in /usr/local/nagios/etc/commands.cfg.

define command {

command_name check_nrpe

command_line $USER1$/check_nrpe -H $HOSTADDRESS$ -c $ARG1$

}

Step3: Test communication with the remote host by using the following command /usr/local/nagios/libexec/check_nrpe -H ip (ip address of remote host on which Nrpe is installed).

This should return the version of Nrpe on the remote host. If the above command does not return the version number make sure the firewall on the remote host does not block the monitoring host.

Step5: Define the service which needs to be monitored on the remote host in the monitoring host.

Example:

```
define service{
use generic-service
host_name remotehost
service_description Swap Usage
check_command check_nrpe!check_users
}
```

Step6: Define the check_users command in the nrpe.cfg file in remote host; this will cause the Nrpe Daemon to run check_users command on the remote host.

command [check_users]=/usr/local/nagios/libexec/check_users -w 10 -c 20.

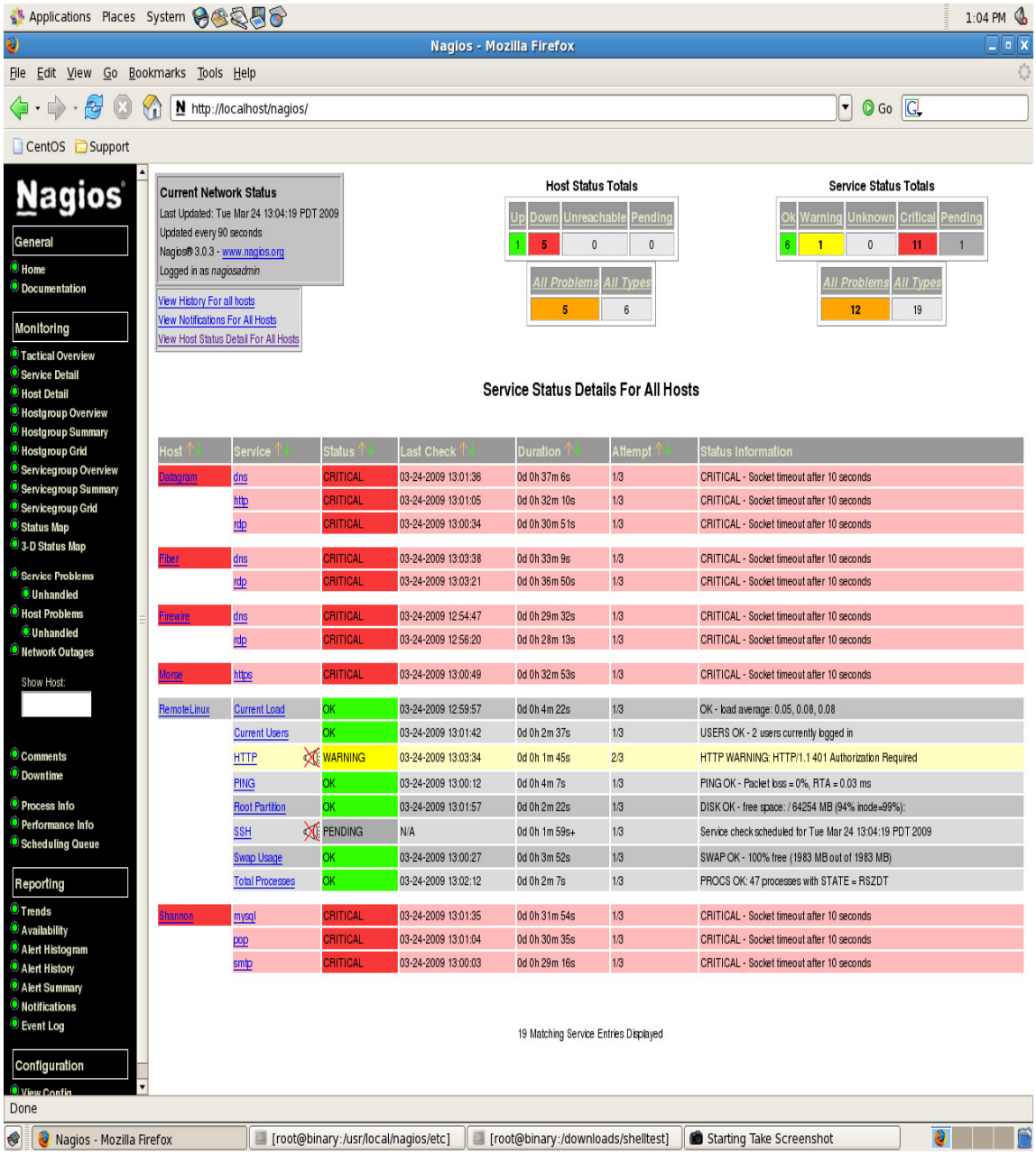


Figure 4.2 Remote Linux Machine Monitoring

The relationship between the agent, plugin, monitoring server and the monitored server in windows environment is depicted in Figure 4.3

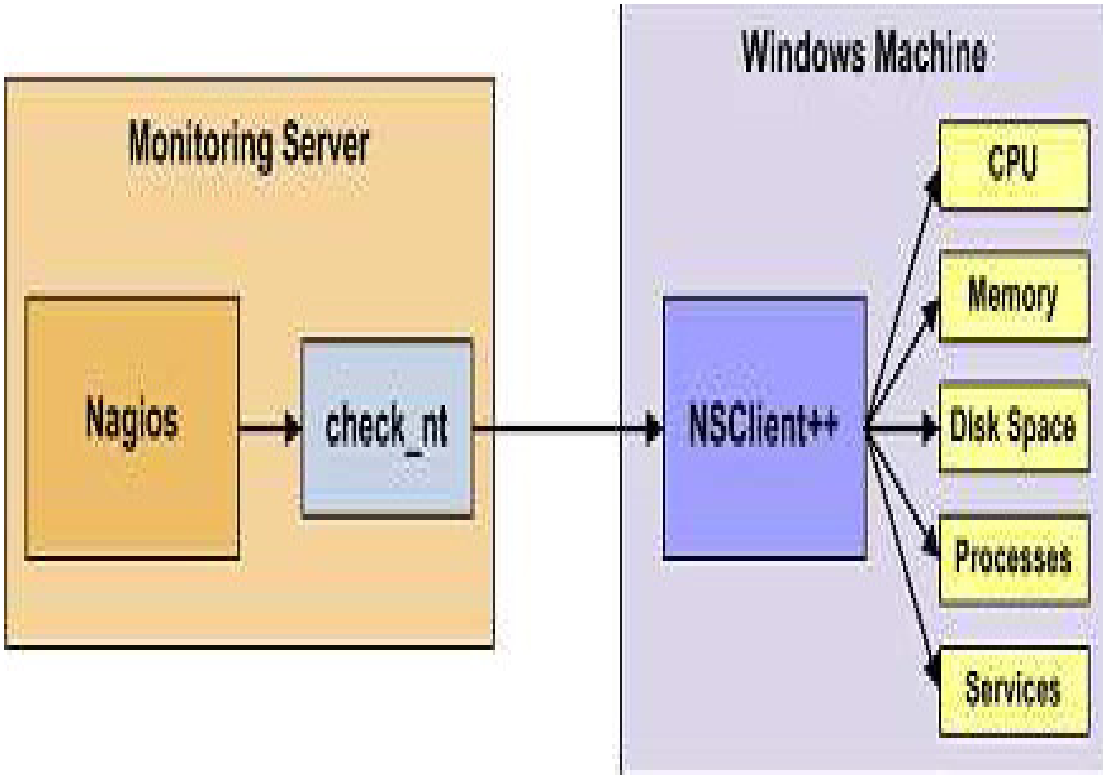


Figure 4.3 Overview of Nagios Passive Monitoring Using NSClient++ (source: www.Nagios.org)

4.2 NSClient++ Implementation Steps [4]:

Step1: Install NSClient++ agent on windows machine (steps to install the NSClient are in the user manual).

Step2: Create new host and service definitions for the remote windows machine where NSClient++ is installed on the nagios machine.

Step3: Specify the command definition for the service defined above at the remote host.

Step4: Include the path where host and services are defined in the nagios.cfg file which is under /usr/local/nagios/etc.

Step5: Save the nagios file and exit.

Step6: Restart nagios with the command service nagios restart.

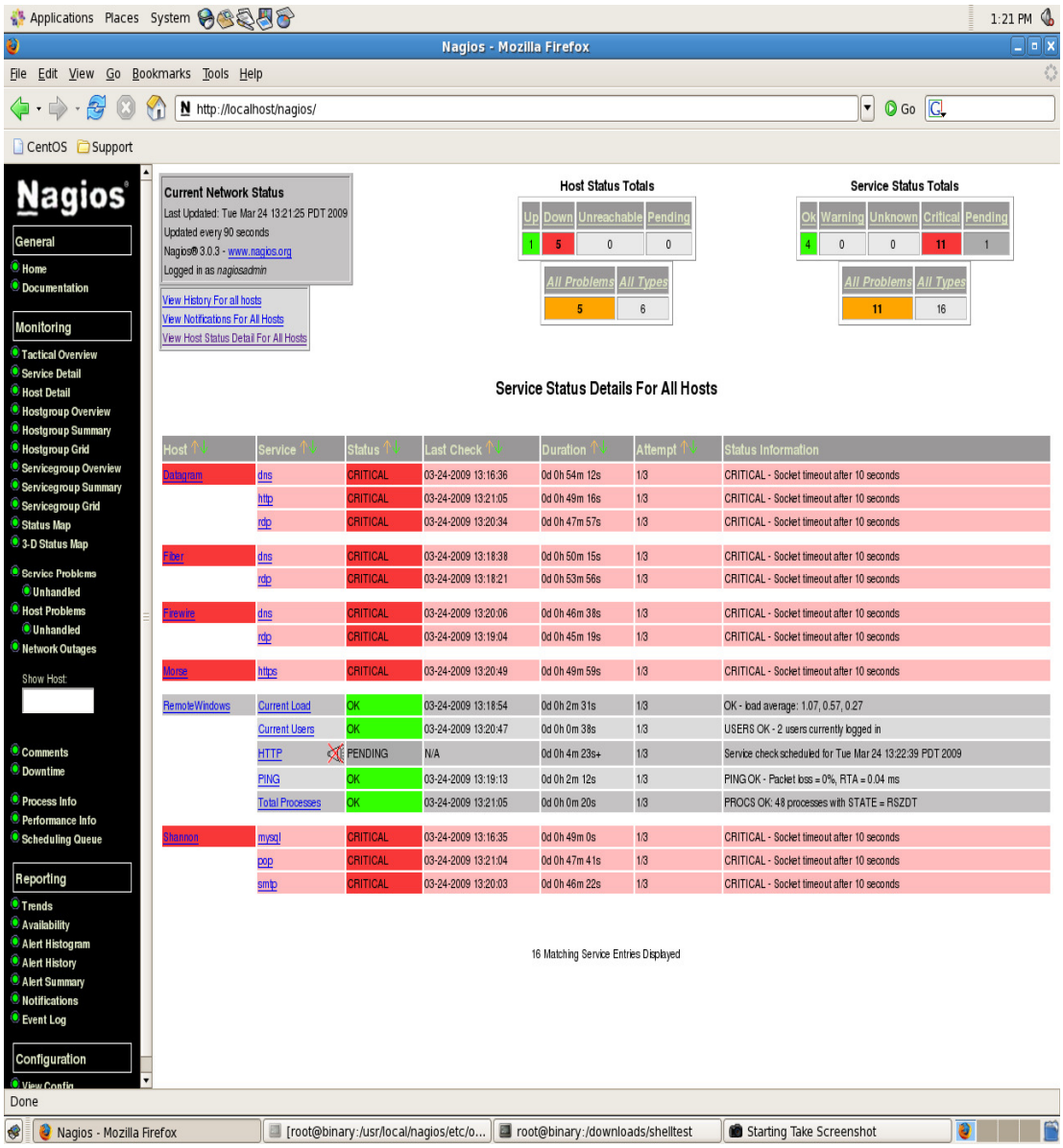


Figure 4.4 Remote Windows Machine Monitoring Using NSClient++

Chapter 5

DISTRIBUTED MONITORING

The reason why we need to have distributed monitoring is to reduce the network traffic. We have increased traffic because of the two Nagios machines doing the same job, that is, both the machines are monitoring the same systems. Therefore, in order to reduce the network traffic we can reduce the overhead on each machine by dividing the work between the two machines, so that a different set of machines are monitored by each server.

The way it is implemented is depicted in Figure5.1; here the system Packet is monitoring systems Shannon, Turing, Seeall and the system Binary is monitoring systems Morse, Smelt, Datagram. This type of monitoring does not perfectly embed itself in our scoring requirement since we have two scoring engines each maintaining their own database. However, in an IT industry the monitoring infrastructure is never centralized, and they do not need a scoring system; in such cases distributed monitoring is done.

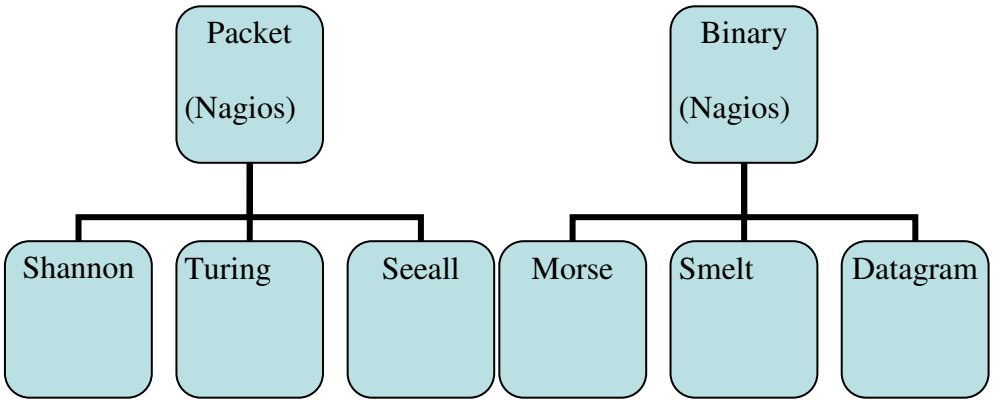


Figure 5.1 Distributed Monitoring

5.1 Distributed Scoring System

However, if distributed scoring needs to be developed, there is a way in which that objective can be attained, i.e. if both the monitoring systems place their monitored data in a separate database and if the scoring is done based on that databases as depicted in Figure 5.2. It's possible to have distributed scoring system.

It was observed in the competition that the defending team blocked the traffic coming from one of the scoring system and allowed the traffic from the other scoring machines, which is the reason that we have different scores from each of the machines. Because we have different sets of data for the same machines which are being monitored, we were able to figure it out, what exactly defending team has done.

The problem with this distributed scoring system is that, if the defending team blocked any of the monitoring system, there is no way for the white team members to know whether the service is down because of the work done by Red team or the defending team have blocked the scoring machine. Because we just have one set of monitored data for each set of machines.

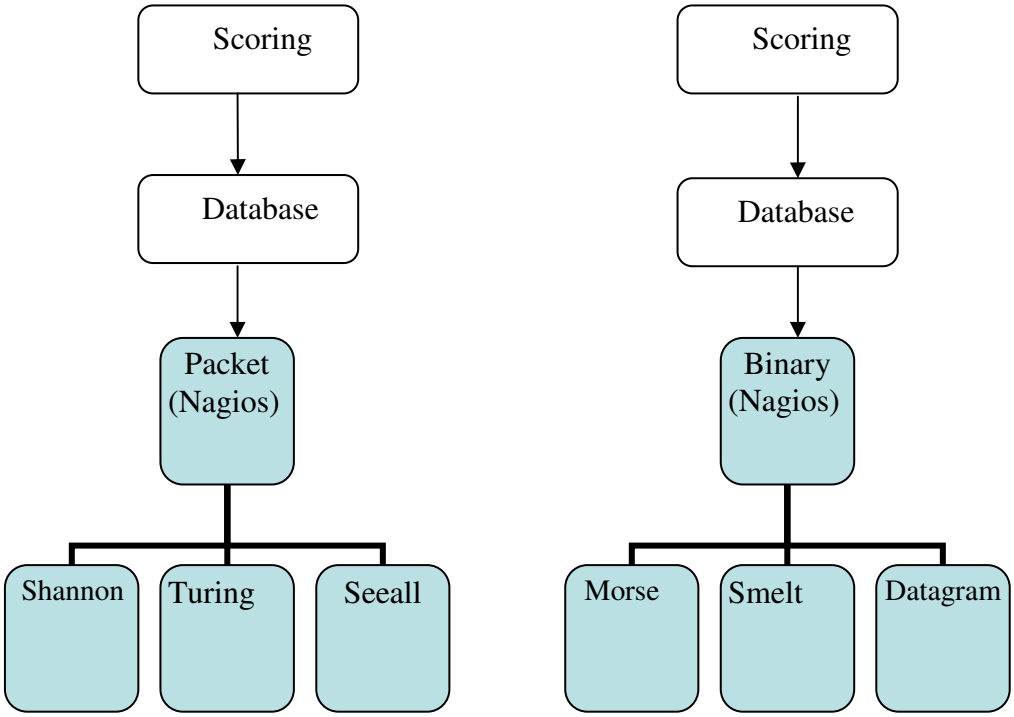


Figure 5.2 Distributed Scoring System

Chapter 6

FAILOVER MONITORING

In college cyber defense competition, we have two monitoring systems doing the same job; we have done that to improve availability. But the problem is, the amount of traffic being generated on the network which might lead to network congestion. One solution to the problem is failover monitoring in which if one system fails, the other system takes the burden. In other words, we establish a master slave relation between the two systems, so that at any point of time we have only one system which monitors the defending teams. We do this by installing a remote agent called Nrpe on the master machine, and the slave keeps on checking the status of Nagios on master machine through check_nrpe plugin which talks to Nrpe on master, so that if master nagios fails, the slave nagios picks up the job of the slave. It does this by enabling the service check options in nagios.cfg file as shown below

“execute service checks =0” to “execute service checks =1” and

“enable notifications=0” to “enable notifications=1”

It can also let someone in white team know that the system has crashed by email, or it can send a message to their cell phone. Although it picks up the job of the master, it still keeps on checking the status of master hoping that white team members would intervene. At some point of time, if the master is up because of intervention of white team members, it goes back to original state. This task is achieved by writing a shell script which checks the condition of master Nagios at regular intervals.

So if at any point, if master is up, the shell script disables the service check in nagios.cfg file. The configuration changes which the shell script does are shown below.

“execute service checks =1” to “execute service checks =0” and

“enable notifications=1” to “enable notifications=0”

It’s also possible for the master two have n number of slaves, such that each monitors a different subset of machines so that if the master fails, the slaves come up. The way it can be implemented is shown in Figure 6.1.

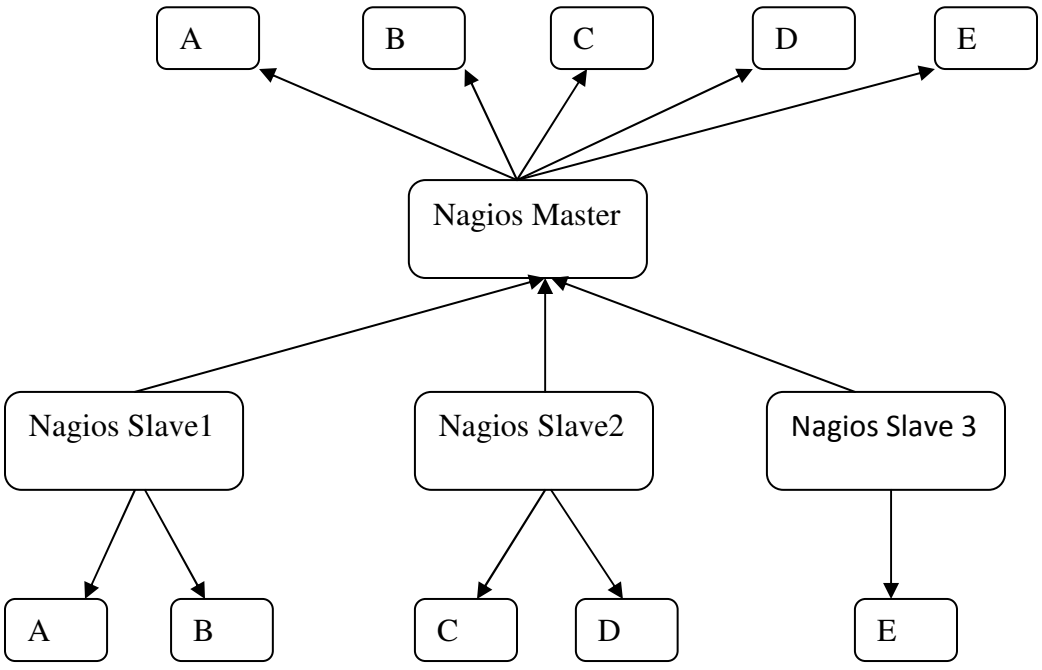


Figure 6.1 Single Master, Multiple Slave Failover Monitoring.

6.1 Failover Monitoring Implementation Steps:

Step1: Install Nrpe agent on the master side

Step2: Add a command definition in /usr/local/nagios/etc/nrpe.cfg of master

command [check_nagios]= -e 1 -F /usr/local/nagios/var/status.log -C /usr/local/nagios/bin/nagios

Where –e is used to indicate the time after which the file is considered stale

-F is used to indicate the file which you are looking for, which is the log file

-C is used to indicate the substring in the process arguments

Step3: Define service check_nagios on the slave side.

```
define service{
        use generic-service
        host_name remotehost
       service_description check nagios
      check_command check_nrpe! check_nagios
                }
```

Step4:check_nrpe plugin by default comes with Nagios but we still need to define in command definition

So add the following definition in /usr/local/nagios/etc/commands.cfg

```
define command {
command_name check_nrpe
command_line $USER1$/check_nrpe -H $HOSTADDRESS$ -c $ARG1$
                }
```

Step5: Test communication with the master by using the following command /usr/local/nagios/libexec/check_nrpe -H ip (ip address of slave on which Nrpe is installed)

This should return the version of Nrpe on the master.

Step6: If the above command does not return the version number make sure the firewall on the master does not block the master

Step7: Write a shell script on slave side that talks with Nrpe on the master side and either enable or disable service checks depending on the status of master.

Step8: Set the shell script as a cron job by editing crontab file which is under /etc/crontab. You can edit this file by the command crontab –e and make sure the script failover.sh is being executed.

Shell script which does the failover monitoring

```
#/bin/sh

declare -a temp

var=`/usr/local/nagios/libexec/check_nrpe -H 172.16.30.229 -c check_nagios`

temp=(`echo $var | tr ':' ' '`)

case ${temp[1]} in

'OK')

echo nagios on PACKET is fine

sed -e 's!execute_service_checks=1!execute_service_checks=0!' -e

's!enable_notifications=1!enable_notifications=0!' /usr/local/nagios/etc/nagios.cfg >

/shelltest/test

cp /shelltest/test /usr/local/nagios/etc/nagios.cfg

service nagios restart

;;

'CRITICAL')

echo nagios on PACKET is critical

sed -e 's!execute_service_checks=0!execute_service_checks=1!' -e

's!enable_notifications=0!enable_notifications=1!' /usr/local/nagios/etc/nagios.cfg >

/shelltest/test

cp /shelltest/test /usr/local/nagios/etc/nagios.cfg

service nagios restart

;;Esac
```

Chapter 7

CONFIGURING NAGIOS

Nagios is configured by building configuration files for each machine that needs to be monitored. This type of configuration is used in places where there are small numbers of systems to be monitored but not in an industry environment because it takes lot of time to build configuration files for each individual machine. The disadvantage of building configuration files is that, a novice cannot configure Nagios unless he has good understanding of the tool.

Steps to be followed for configuration:

Step1: Identify the systems to be monitored.

Step2: Identify the IPs of each machine since nagios monitoring is IP based.

Step3: Identify the services running on each machine.

Step4: Prepare configuration files for each machine which defines host and service definition and name each file with .cfg extension.

Step5: Define command definition for each service of the configuration file in command.cfg file.

Steps6: Include the path where these configuration file are present in nagios.cfg because when nagios starts it reads this file.

Step7: Since there are two defender teams it’s difficult to distinguish the systems of each group so in order to identify them create two host groups.

Step8: Start nagios with the command “service nagios start”.

Configuration of CCDC monitoring system:

Step1: Identify the systems to be monitored

Step2: Identify the IP address of each machine.

Step3: Identify the services running on each machine.

Step4: Here is how a configuration file in nagios is built say for monitoring http on a machine called Datagram. Create a file called Datagram.cfg and define the host and service definition for that machine as follows.

```
#definition for Datagram machine
define host{
        use         ccdc-host
        host_name  Datagram
        alias       Datagram
        address     192.168.129.117
       }
define service{
          use ccdc-service
          host_name Datagram
          service_description    DNS
          check_command check_dns!smelt.local!192.168.66.210
          }
```

Step5: For the service defined above in step4 you need to define a command in commands.cfg which is in /usr/local/nagios/etc directory.

```
#'check_dns'
define command{
command_name check_dns
command_line $USER1$/check_dns  -s $HOSTADRESS$  -H $ARG1$ -a $ARG2$
                }
```

Where

-s: specifies the DNS server where the lookup needs to be done.

-H: specifies the name of the machine which is being searched.

-a: specifies the expected ip which the DNS server needs to return.

Step6: Open the nagios.cfg file which is usually in /usr/local/nagios/etc/objects/nagios.cfg and place the path of configuration file in this file.

Step7: Since there are two teams create groups which consists of machines from each team, since datagram is from say red1 declare the below definition in datagram.cfg

```
define hostgroup{
    hostgroup_name  Red1;
    alias           Red1; Long name of the group
    members         Datagram,Smelt,Turing,seeall;
    }
```

The other service and command definitions to monitor each service is show below.

HTTP:

Service definition:

```
define service{
        use ccdc-service
        host_name Smelt
        service_description    HTTP
        check_command check_http!”33d!”It works”
        }
```

Command definition:

```
define command{
        command_name check_http
        command_line  $USER1$/check_http  –I  $HOSTADDRESS$  -M  $ARG1$  -s
        $ARG2$
                }
```

Where

-I: specifies the host address

-M: specifies the days after which the document is considered old.

-s: specifies the string which needs to be found on the page

HTTPS:

Service definition:

```
define service{
        use ccdc-service
        host_name Michael
        service_description    HTTPS
        check_command check_https!443!http://192.168.194.210/index.php
        }
```

Command definition:

```
define command{
        command_name check_http
        command_line   $USER1$/check_http  –I  $HOSTADDRESS$  -p  $ARG1$  -S
        -u $ARG2$
                }
```

Where

-I: specifies the host address.

-p: specifies the port number.

-S: specifies SSL connection.

-u: specifies the URL.

SMTP:

Service definition:

```
define service{
        use ccdc-service
        host_name  Dwight
        service_description    SMTP
        check_command          check_smtp!HELO          from          white!"tech-
com.cyberdyne.local"!5!10
        }
```

Command definition

```
define command{
      command_name check_smtp
      command_line  $USER1$/check_http  –H  $HOSTADDRESS$  -C  $ARG1$  -R
       $ARG2$ -w $ARG3$ -c $ARG4$
              }
```

Where:

-H: specifies the host address.

-C: specifies the command which needs to be executed at the remote server.

-R: specifies the response which is expected for the command which is specified by option –C.

-w: specifies time after a warning should be given.

-c: specifies time after which a critical message should be given.

POP:

Service definition:

```
define service{
        use ccdc-service
         host_name  Dwight
        service_description    pop
      check_command check_pop!110!Dovecot
            }
```

Command definition:

```
define command{
         command_name check_pop
         command_line  $USER1$/check_pop  –H  $HOSTADDRESS$  -p  $ARG1$  -e
          $ARG2$
                    }
```

Where:

-H: specifies the host address.

-p: specifies the port number.

-e: specifies the expected string in the response.

Security is another major strength of Nagios which uses SSH protocol, which sends monitoring data in encrypted format across the network. Here is an example of how SSH can be used to monitor SMTP (simple mail transfer protocol) status using SSH. The

service SMTP needs to be defined in the configuration file of the machine which needs to be monitored.

```
define service{
                    use ccdc-service
                    host_name Fiber
                   service_description smtp
                     check_command check_smtp
                     }
```

For the above service the command definition needs to be declared in the command.cfg file in the below way using check_by_ssh plug-in.

```
define command{
   command_name check_smtp
command_line $USER1$/check_by_ssh -H $HOSTADDRESS$ -C check_smtp
                    }
```

Chapter 8

AUTOCONFIGURATION

The Reason why we have Autoconfiguration is because the earlier discussed configuration in chapter seven is not practical in an IT industry when there are hundreds of systems to be monitored, as it takes lot of time to build configuration files for each system. Instead of that, we can have a text file in particular format, which specifies the machine name, the machines IP and the services to be monitored. We can have a shell script, which takes text file as a feed and generates configuration files depending on the text file and keeps the generated file paths in nagios main configuration file so that when Nagios starts, it starts looking for these machines based on their IP and consequently checks for their services on the particular port.

8.1 Autoconfiguration Using TCP/IP Handshake

Here are the steps for Autoconfiguration which configures Nagios, so that Nagios uses TCP/IP handshake to know the service is running or not. Steps for Autoconfiguration to do stringent service checks are specified in later part of this chapter.

Stage1: A shell script which installs nagios.

The user needs to be sure that “yum”package installed on the system .Once the user is sure he has “yum”installed, the user needs to copy the folder called CCDC from the flash drive under root, and the user needs to be in ccdc directory before he runs the script. Then run the script called nagios-installation.sh so that it’s going to install the necessary packages required for Nagios, which are glibc-common, gcc, and gd-devel.

During the installation the system asks the user for an email to whom notifications need to be sent, this generally refers to the Nagios administrators email. The script is written in such a way that if any module installation fails, the script will no longer continue to execute and it comes out of the installation. At the same time the script generates a log file called nagiosinstallation.log and any errors in the installation can be verified by viewing this log file.

In the CCDC competition when we have a 24*7 environment we don't have any problems with it, but when we have a holiday in the middle of the competition, and if we have Nagios running, it's going to affect the scoring system calculating points even when not required. So instead of manually stopping and starting Nagios, you can schedule it to start and stop before the competition begins by modifying the time templates.cfg file in objects folder of Nagios.

All you have to do is to make sure that us-holidays definition has the holidays declared for example say November 11th. And there is one more definition called 24*7_sans_ holidays which uses us-holidays definition, the main objective of this time period definition is to work on the scheduled time periods except for the scheduled downtime in us-holidays definition. Then you need to make sure the service definition and host definition and contact information definition uses the 24x7_sans_holidays.

After Stage 1 you have the local host configured to be monitored by Nagios. If you start Nagios with the command "service nagios start," and if open your browser with the address http://localhost/nagios, you can see the following page shown in Figure 8.1 where

you are supposed to enter the Nagios admin password which you entered during installation in Stage1.

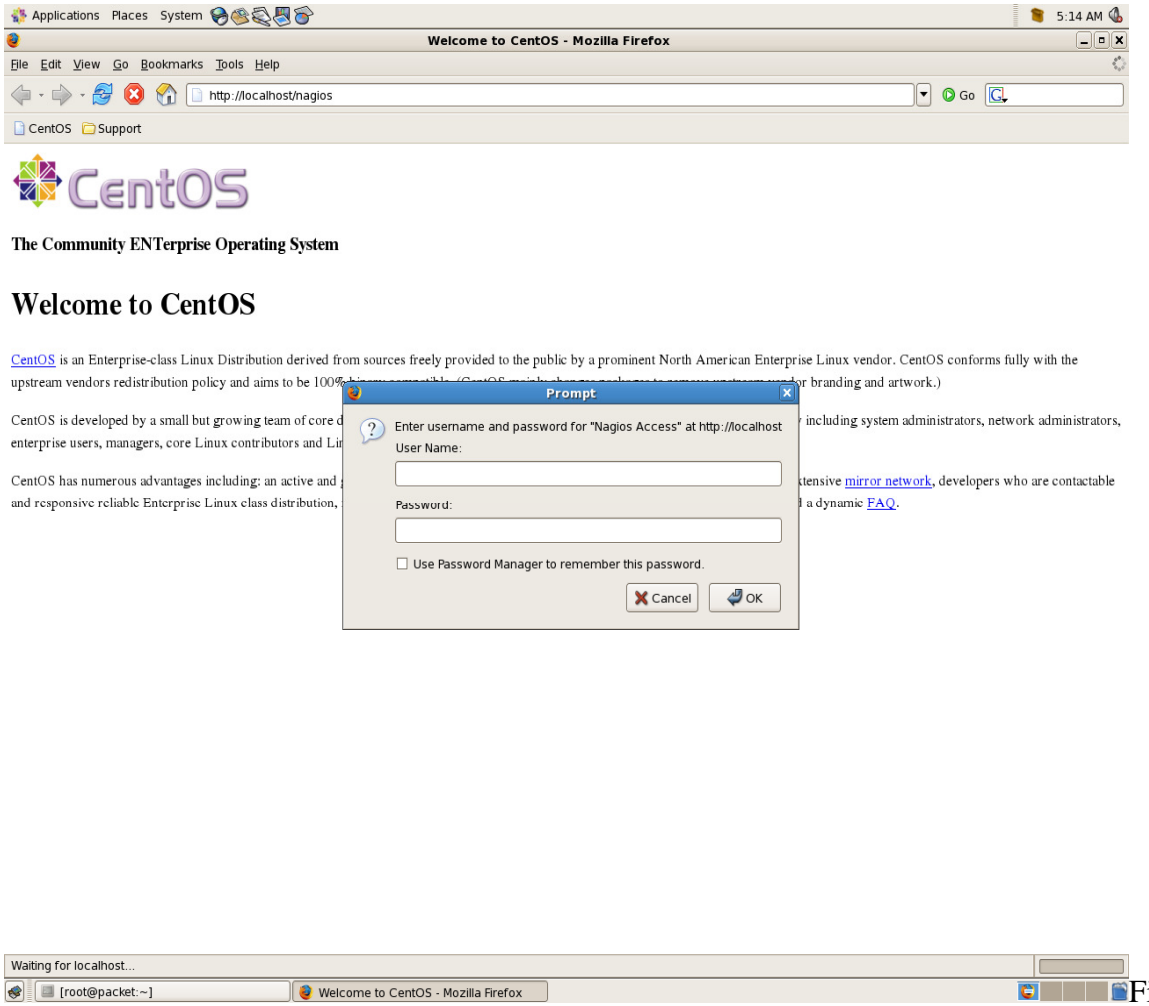


Figure 8.1 Nagios Login Page

Nagios by default configures the local host with 127.0.0.1 which is the loop back address of the machine on which Nagios is installed, this can be seen in Figure 8.2.

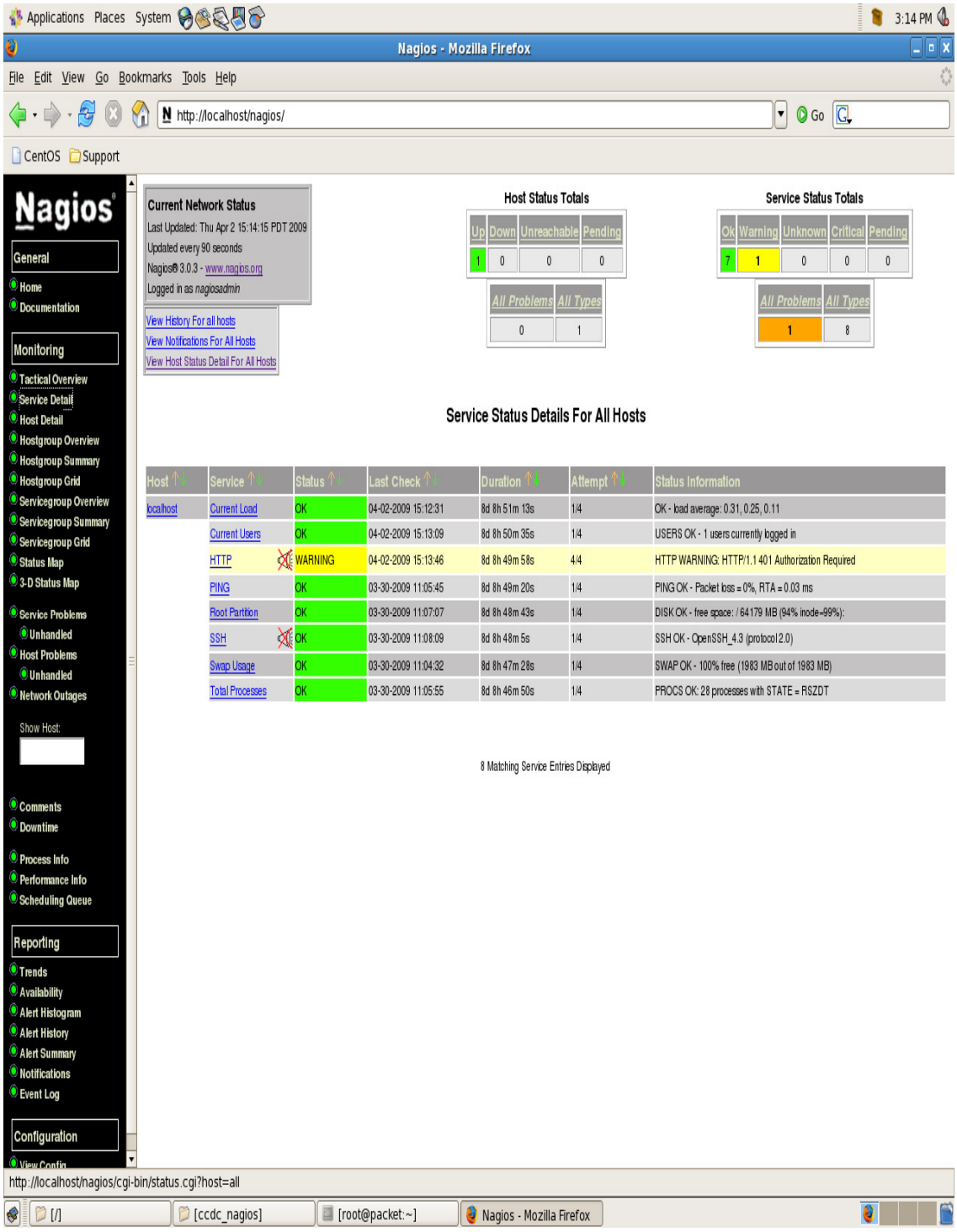


Figure 8.2 Local Host 127.0.0.1

At this point you can even open the email which you specified in stage 1 and see the notifications.

Stage2: The second script is the nagios-config-files.sh script: This main functionality of this script can be described in two stages.

Stage2.1: Script generates ccdc host, ccdc service, and contact templates and places them in templates.cfg file which are later used by host and service definition. The parameters which are configured in the templates are as follows.

Active_checks_enabled: Setting this to 1 allows Nagios directly to monitor service on the remote machine. This was specified as 1.

Passive_checks_enabled: Setting this to 1 allows Nagios to get service data from a remote machine such as Nrpe. This was specified as 1.

Contact_groups: This is by default is specified as admins and the email address specified by the user during installation in stage 1 refers to this particular contact.

Notifications_enabled: Allows Nagios to send notifications to the email specified during stage1. This was specified as 1.

Notification_options: It depends on nagios administrators requirement whether he want to be notified for only critical messages then it should be specified as ‘c’ if he wants to be notified for warning it should be specified as ‘w’, if he wants to be notified for unknown states then it should be declared as ‘u’.if he wants to be notified for all the states then it should be specified as ‘w, c, u’.

Notification interval: This specifies the time gap between two notifications. This was specified as 60, so notifications will be sent for every hour.

Notification period: This refers to the time template during which he likes to receive notifications. This was specified as 24*7, so that notifications could be sent during any time of the day.

Service_notification_commands: & Host_notification_commands:

This specifies the way in which the notification needs to be sent since we are using email We specify this by the command 'notify-service-by-email'.

Max_check_attempts: This specifies the maximum number of times nagios should check for a service or host state to confirm a state. This was specified as 3 attempts.

Normal_check_interval: This specifies the time interval between two attempts, to be clear if it finds a hard state like 'ok' or critical in the first attempt, the time gap for the next check is specified by this option. This was specified as 5 minutes.

Retry_check_interval: If nagios finds the service is unknown then this particular option specifies when the next check should be done instead of normal_check_interval. This was specified as 1 minute.

Stage2.2: Script defines host and services of each machine that needs to be monitored and nagios-config-text serves as input for this script, so before the user runs this script he needs to make sure that nagios-config-text is written.

Here is how nagios-config-text needs to be written

Machine ip, machine name, service 1, service 2, service 3

Example:
192.168.129.117, Datagram, http, rdp, dns
192.168.130.112, Shannon, smtp, pop, mysql

Once this text file is written run the configuration script (nagios-config-files.sh):

Example of Configuration text

192.168.130.114,Fiber,dns,rdp

192.168.130.117,Firewire,dns,http,rdp

192.168.130.110, Morse, https

So for example for the first line it creates a file called Fiber.cfg in usr/local/nagios/etc/objects with the following content

```
define host{
        use       ccdc-host
        host_name  Fiber
        alias      Fiber
        address    192.168.130.114
       }
define service{
         use ccdc-service
         host_name Fiber
         service_description  dns
         check_command check_dns
```

Uses ccdc-host template which is in templates.cfg created by stage 2.1

Uses ccdc-service template which is in templates.cfg created by stage 2.1

}

And in the same way it generates configuration files for each machine depending on number of hosts in the text file.

Stage3:

A shell script (nagios-commands.sh) which defines command definition (commands trigger TCP check with each port) for each service defined in stage 2 based on the input from a text file called nagios-commands-text. The file nagios-commands-text should be written in the following format before executing the shell script, machine IP, machine name, service being checked, port number of the service being checked. Once this text file is written, run the script called nagios-commands.sh.

Example of command_text:

192.168.130.114, Fiber, dns, 53, rdp, 3389

192.168.130.117, Firewire, dns, 53, rdp, 3389

192.168.130.110, Morse, secht, 443, http, 80

192.168.129.117, Datagram, rdp, 3389, dns, 53

So for example for the first line it generates the following command definition in /usr/local/nagios/etc/objects/commands.cfg

```
 #check_dns using TCP
define command{
        command_name check_dns'
```

```
        command_line $USER1$/check_tcp -H $HOSTADDRESS$ -p 53
        }
#check_rdp using TCP
define command{
        command_name check_rdp'
        command_line $USER1$/check_tcp -H $HOSTADDRESS$ -p 3389
        }
```

Once the above command definitions are generated if you restart Nagios and open your browser with http://localhost/nagios you can see all the changes you made in the GUI. The different hosts along with their services can be seen in Figure 8.3.

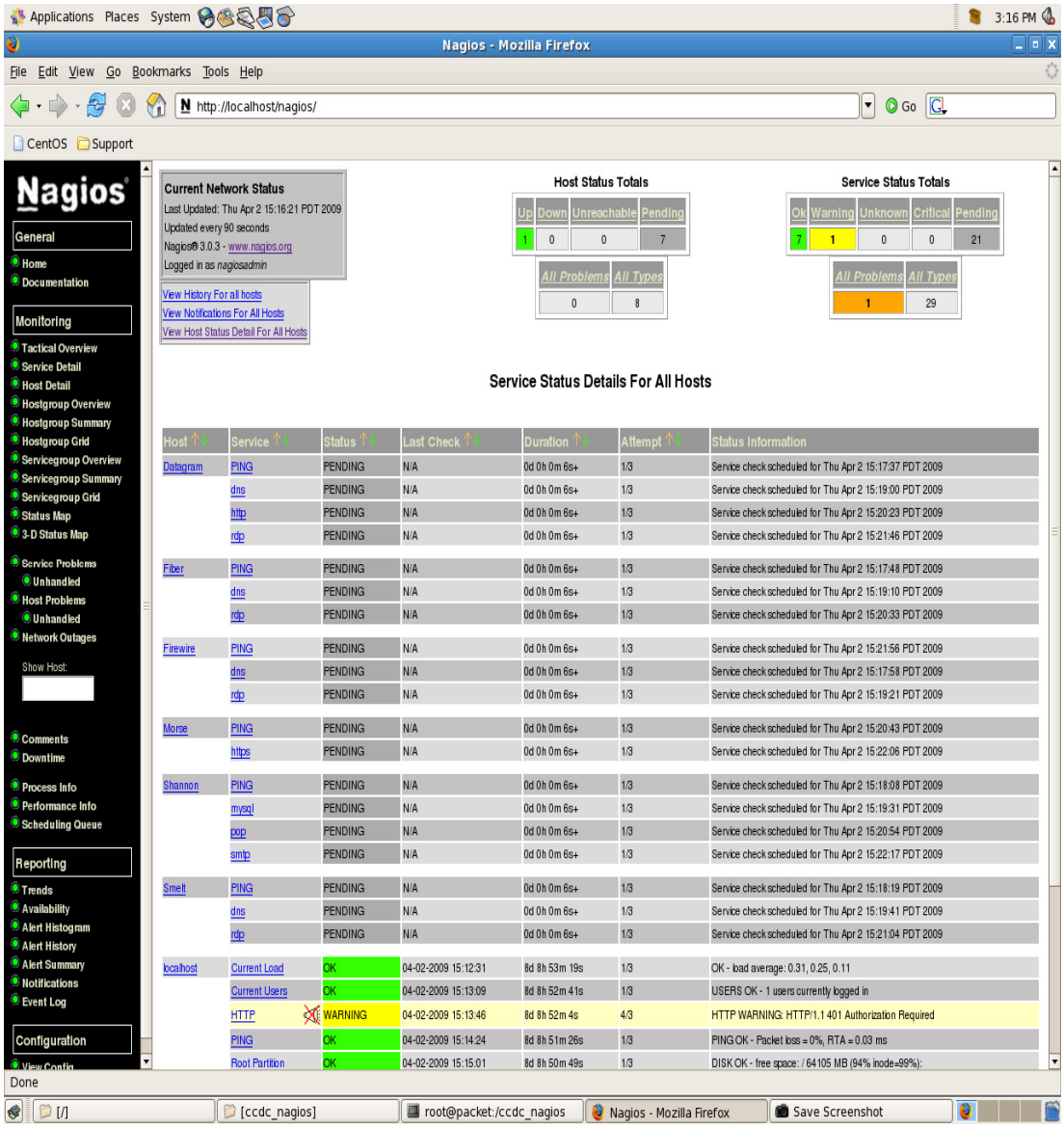


Figure 8.3 Third Stage of Autoconfiguration.

Stage4: In the competition, since there are two groups, it is difficult to distinguish the groups from a single snap shot of Nagios, but Nagios allows creating host groups so that each group can be distinguished. So declare host groups definition on any one machine from each group; this host group declaration is also automated in the project all that user needs to do is write a text file called hostgroup_text in the following format, group name, machine1,machine2 each separated by comma, the second group beginning from the next line. Once this script is written run the script called hostgroup.sh.

Example of hostgroup_text:

Red1, Morse, Binary

Red2, Shannon, Smelt, Packet, Datagram

So for example for the first line it generates the following definition in morse.cfg file which is under /usr/local/nagios/etc/objects

```
 define hostgroup{
       hostgroup_name  Red1
       alias Red1  ; Long name of the group
       members  Morse,Binary;
                              }
```

So for the next line it creates the below hostgroup definition in Shannon.cfg

```
define hostgroup{
        hostgroup_name Red2
        alias Red2  ; Long name of the group
        members Shannon, Smelt, Packet, Datagram;
                                }
```

So once the above definition is defined, if you restart nagios and click on the hostgroup summary radio button, you can see the host and status information of each group in two different columns as seen in Figure 8.4.

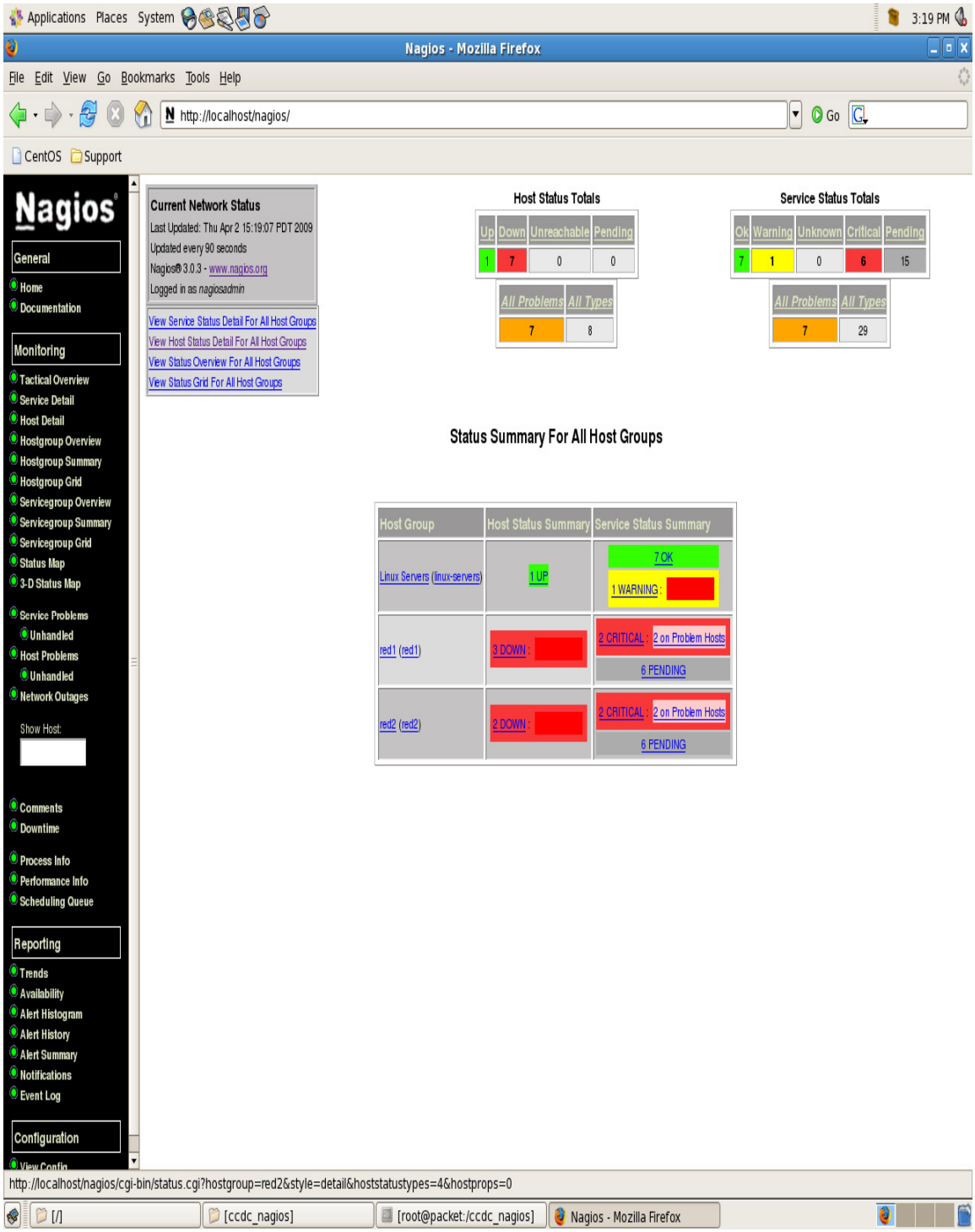


Figure 8.4 Host Groups

Stage5:

Do a preflight check by running script called preflight.sh. To just check things are fine if there are errors check you nagios configuration.

The order of the scripts which needs to be executed for Autoconfiguration is shown in the Figure 8.5

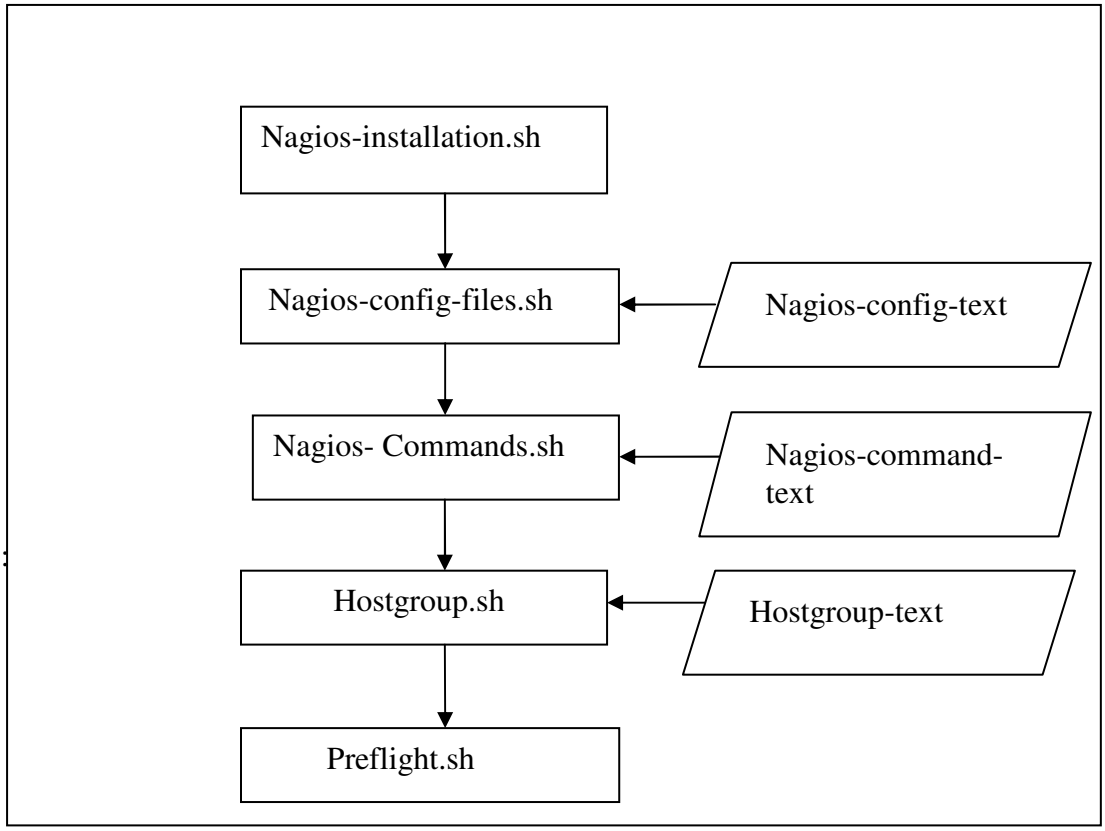


Figure 8.5 Sequence of Steps for Autoconfiguration.

8.2 Autoconfiguration for Stringent Service Checks*:*

Sometimes a TCP/IP handshake is just not sufficient to know the functionality of a service on a remote machine, in such cases we check each service in a stringent way. The different steps to be followed are specified below and they are also shown in Figure 8.6

Step1: Execute Nagios-installation.sh script

Step2: Make sure nagios-config-text file is written in the following format

ip, machine name, and list of services separated by comma.

Example: 192.168.130.114, Fiber, dns, rdp

Step3: Execute nagios-config-gen.sh script which expects input from the user, and it then defines each service according to the input given by the user.

Step4: Execute nagios-commands-gen.sh script which defines the commands in commands.cfg file for the services specified in nagios-config-text file.

Step5: Make sure hostgroup-text file is written in the following format, Group name, and list of members separated by commas.

Example: Red2, Shannon, Smelt, Packet, Datagram

Step6: Execute hostgroup.sh script which generates host group definition.

Step7: Execute preflight.sh script to check whether there are errors in the configuration.

Step8: Start the service with the command service nagios start.

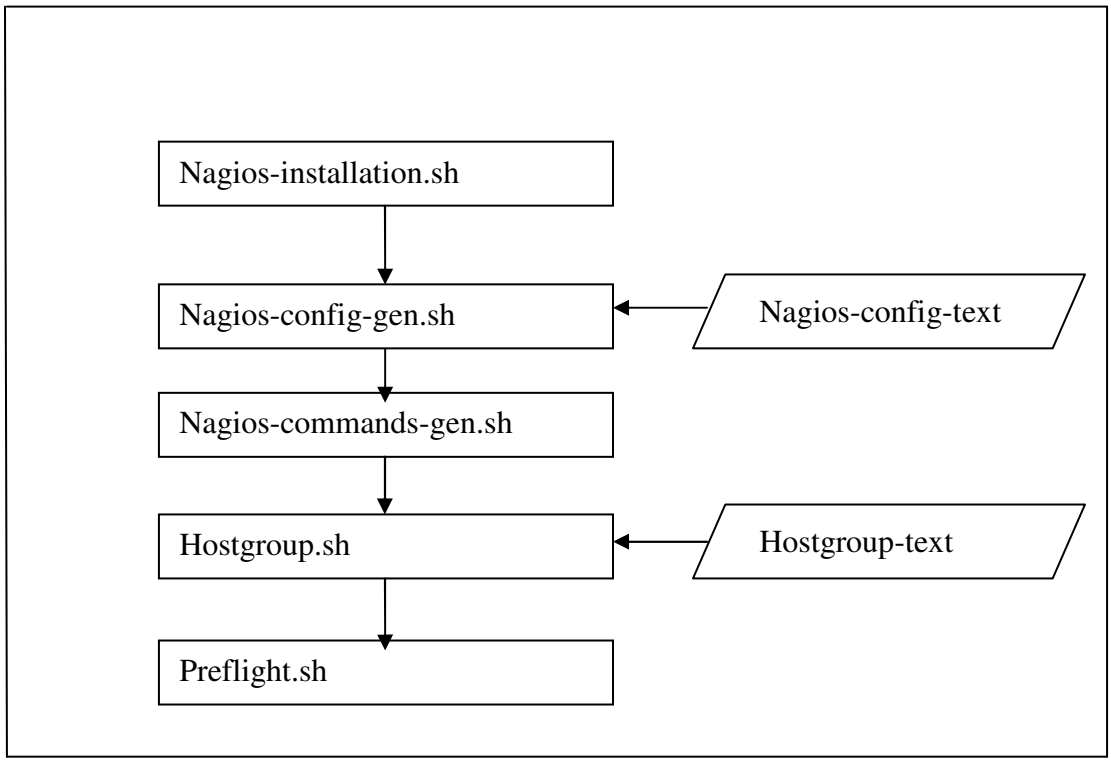


Figure 8.6 Sequence of Steps for Autoconfiguration to Do Stringent Service Checks.

Chapter 9

BUILDING PLUGINS

Before we write plugins, we need to understand what a plugin is? A plugin is a piece of code, either written in Shell or Perl. Nagios already has inbuilt plugins but there is always a requirement that's one requirement is not met with the plugins that comes with Nagios.

For example the plugins which come with nagios to monitor Mysql database are check_mysql and check_mysql_query, cannot insert a row or delete a row in the database, to know the status of the database such actions needs to be done. In such cases we write our own scripts, and configure Nagios so that Nagios is going to poll the script, and uses the results reported by the script to determine the status of service. Here is a plugin which monitors the status of Mysql database by inserting a row and then selecting it, there after deleting the row earlier inserted and then selecting the rows in a table called student in test database.

9.1 Steps to Monitor Mysql Database

Step1: login as root user in the local Mysql database called test.

Step2: Create a table called student with just one row and one column called "id" in database test.

Step3: Create a user called "me" with password called "pass".

Step4: grant select, insert, delete operation privileges on this table for this user called "me" with the command.

GRANT select, insert, delete ON test.student TO me;

Step5: Write a shell script which inserts a row, and selects it, deletes a row which got inserted earlier and then selects it.

The plugin check_db uses a header called jutils.sh where the states are defined.

Below is the jutils.sh script

```
#! /bin/sh

STATE_OK=0
STATE_WARNING=1
STATE_CRITICAL=2
STATE_UNKNOWN=3


if test -x /usr/bin/printf; then
	ECHO=/usr/bin/printf
else
	ECHO=echo
fi

print_revision() {
	echo "$1 v$2 "
	$ECHO "The nagios plugins come with ABSOLUTELY NO WARRANTY. You may redistribute\ncopies of the plugins under the terms of the GNU General Public License.\n For more information about these matters, see the file named COPYING.\n" | sed -e 's/\n/ /g'
}

support() {
	$ECHO "Send email to vasireddy.jaipal@gmail.com if you have questions\n regarding use of this plugin. " | sed -e 's/\n/ /g'
		}
```

Here is the check_db plugin which inserts a row, selects it, deletes a row and then selects it.

```
#!/bin/sh
PROGNAME=`/bin/basename $0`
PROGPATH=`echo $0 | sed -e 's,[\\/][^\\/][^\\/]*$,,'`
. $PROGPATH/jutils.sh

print_usage() {
   echo "Usage: $PROGNAME -H hostname"
   echo "Usage: $PROGNAME --help"
                }

print_help() {
   print_usage
   echo ""
   echo "Inserts a row selects it, deletes a row and selects it from a table called dbmonitor
inside database test"
   echo ""
   support
}

# Make sure the correct number of command line
# arguments have been supplied if you want to monitor remote database

#if [ $# -lt 1 ]; then
#    print_usage
#    exit $STATE_UNKNOWN
#fi
# Get the command line arguments
#host=$2
#echo $host

## If the host name doesn't exist, exit
#
#if [ ! -e $host ]; then
#    $echo " could not find host ip!\n"
#    exit $STATE_UNKNOWN

IRESULT=`/usr/bin/mysql --user=me --password=pass <<SQL
use test
insert into dbmonitor values (1);
```

```
select * from dbmonitor;
quit
SQL`
case "$IRESULT" in
*1)
IRESULT=1;
;;
esac
DRESULT=`/usr/bin/mysql --user=me --password=pass <<SQL
use test
delete from dbmonitor where id=1;
select * from dbmonitor;
quit
SQL`
LEN=$(echo ${#DRESULT})
if [ $LEN -eq 0 ];then
DRESULT=0;
fi
exitstatus=$STATE_WARNING #default
if [ $IRESULT -eq 1 ];then
if [ $DRESULT -eq 0 ];then
echo check_db:OK INSERT AND DELETE SUCCEFULL
exit $STATE_OK
else
echo  check_db:CRITICAL INSERT SUCCESSFULL BUT DELETE FAILURE
exit $STATE_CRITICAL
fi
else
echo check_db:CRITICAL INSERT FAILURE
exit $STATE_CRITICAL
fi
exit $exitstatus
```

Step6: We integrate the plugin into nagios by defining a service in the configuration file of the local machine, and defining a command in the command definition in localhost.cfg

```
define service{
   use local-service
   host_name localhost
   service_description  DBcheck
   check_command check_db
   }
```

We specify the exact path where the script is located in command.cfg.

```
define command{
   command_name check_db
   command_line /ccdc_nagios/nagios-plugins-1.4.13/plugins-scripts/check_db
  }
```

Step7: Restart Nagios with the command “service nagios restart”. Hers is a screenshot of database monitoring on local host in Figure 9.1.

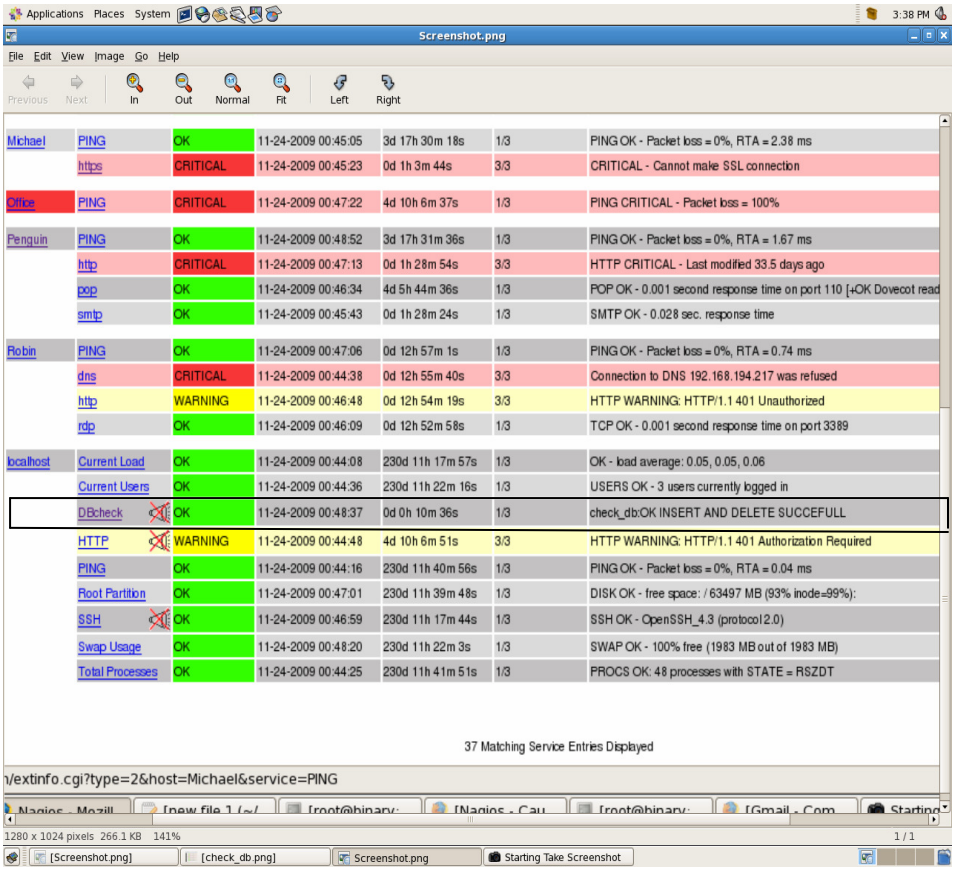


Figure 9.1 Database Monitoring Using check_db Plugin.

Chapter 10

INJECT AUTOMATION

In the beginning of the competition, we have someone among the white team, who is sending injects for the defending team at a particular point of time. The white team members need to go to the machine, to check whether the job has been done or not. In order to reduce the burden on the white team, even these injects can be automated by using shell scripting. Here we build a text file specifying the name of the day, month, the time at which the inject needs to be send, the time from which Nagios starts looking for that particular service, the machine name on which the service needs to be checked, the name of the service and the port number on which Nagios starts checking for a hand shake.

We have a shell script which is being executed as a cron job which takes this text file as input and processes the date command and looks for a matching date and time in the text file. If it matches the date and time in the text file, it immediately sends an email to the entire defending team saying that a service needs to be installed at particular port. If time matches the Nagios checking time, it immediately changes the configuration file of that machine on which the service needs to be installed and restarts Nagios so that changes are going to take effect and Nagios starts looking for this new service. The functionality of the inject automation script is depicted in Figure 9.1.

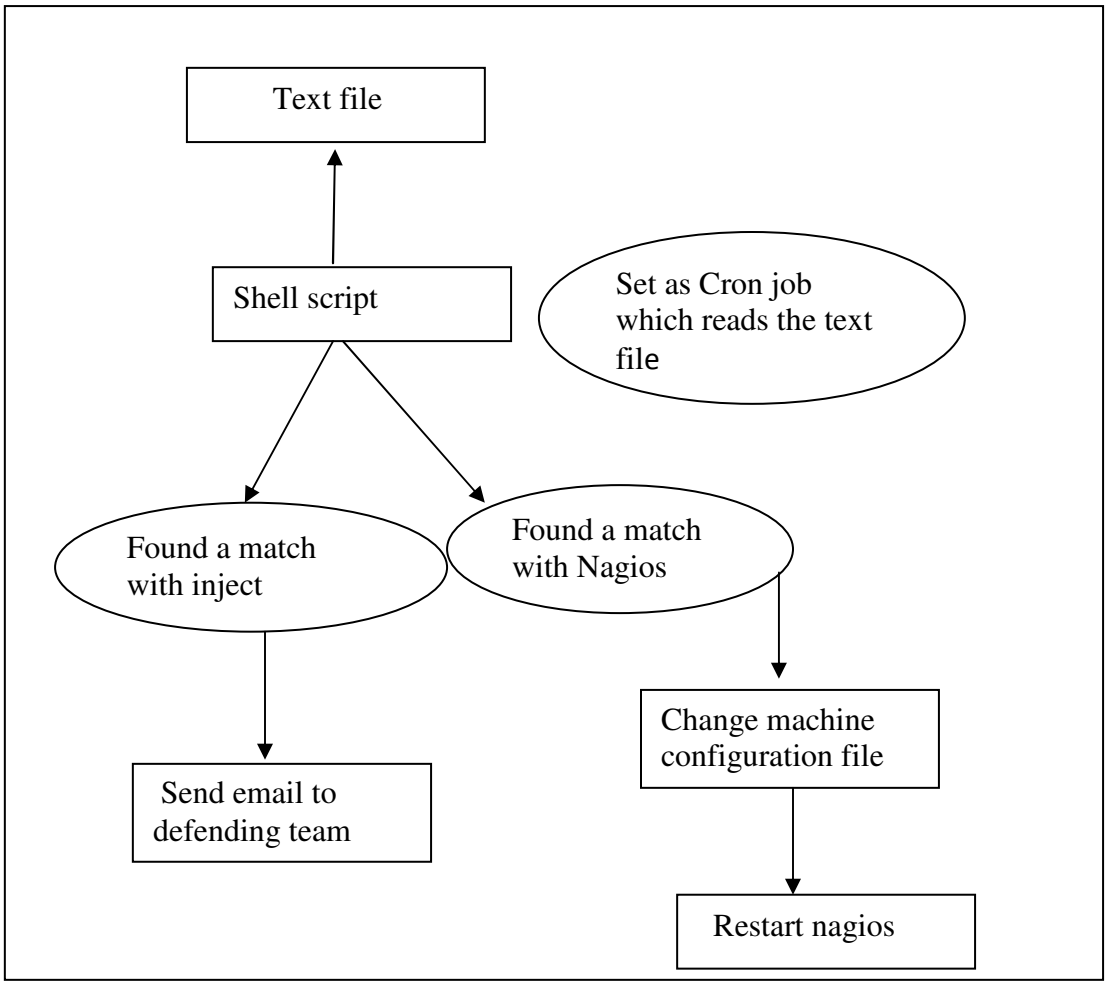
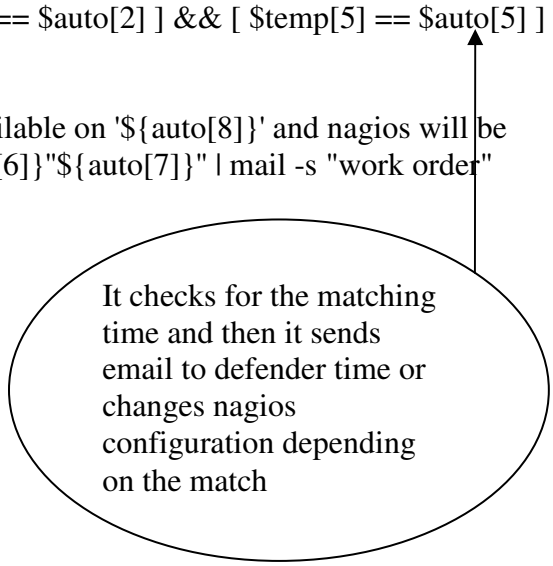


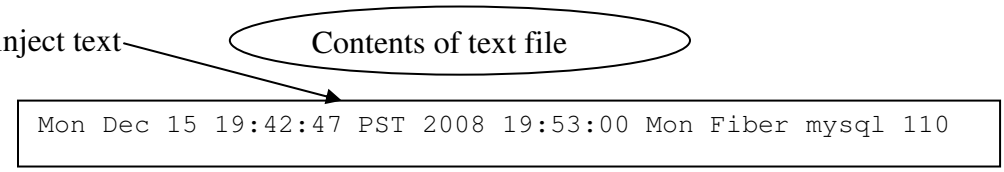
Figure 10.1 Functionality of Inject Automation Script

Here is the shell script which reads a text file

```
#/bin/sh
#the text file which this shell script should read should be in the following format
example Mon Dec 15 19:42:47 PST 2008 19:53:00 Fiber mysql 53
#where 19:42:47 is the time when the email will be sent to the blue team #regarding the
inject and at 19:53:00 it automatically configures nagios for #monitoring
declare -a b d
var=`date`
temp=(`echo $var | tr ',' ' '`)
a=${temp[3]}
echo $a
while read line ;do
auto=(`echo $line | tr ',' ' '`)
b=${auto[3]}
c=${auto[6]}
if [ $temp[1] == $auto[1] ] && [ $temp[2] == $auto[2] ] && [ $temp[5] == $auto[5] ]
&& [ $a == $b ]
then
echo 'the service '${auto[9]}' should be available on '${auto[8]}' and nagios will be
checking on port ‘${auto[10]}'  from {auto[6]}"${auto[7]}" | mail -s "work order"
defender@ecs.csus.edu
echo same time
else
echo its a different time
fi
if [ $a == $c ] && [ $temp[0] == $auto[7] ]
then
echo same time
echo 'define service{
        use ccdc-service
        host_name '${auto[8]}'
        service_description  '${auto[9]}'
        check_command check_'${auto[9]}'
        }'>> /usr/local/nagios/etc/objects/${auto[8]}.cfg
if cat /usr/local/nagios/etc/objects/commands.cfg | grep ${auto[9]}
then
echo present
else
echo '#check_'${auto[9]}'
define command{
        command_name check_'${auto[9]}'
        command_line $USER1$/check_tcp -H $HOSTADDRESS$ -p '${auto[10]}'
```



```
        }'>> /usr/local/nagios/etc/objects/commands.cfg
fi
service nagios restart
fi
done < inject text
```



So for the above text file at 19:42:47 PST 2008, the machine sends an email to defending team with the subject “work order” saying that 'the service Mysql should be available on fiber and Nagios will be checking on port 110 from 19:53:00 Mon.

At 19:53:00 it adds the below definition to the configuration file Fiber.cfg which is under /usr/local/nagios/etc/objects.

```
define service{
        use ccdc-service
        host_name Fiber
        service_description mysql
        check_command check_mysql
            }
```

It then checks whether Mysql definition is already present because it might be defined as a service on other machines, so if the definition does not exist, it adds the command definition in /usr/local/nagios/etc/objects/commands.cfg file and then it restarts Nagios so that Nagios starts monitoring Mysql on Fiber at port 110.

The only problem with this automation is performance, which gets reduced because of the date command in the shell script, which gets executed every minute; the other way is

to edit the crontab file which is under /etc.you can edit this file by the command crontab -e.

The advantage is that you already have the UNIX operating system, which takes care of the date. So the only thing which we need to do is prepare injects, identify the path where injects are located and identify the team to whom injects need to be mailed. This can be better explained with an example say we have two injects which needs to be mailed on April 8th at 2 p.m and another one at April 11th at 10 a.m6 to Team@ecs.csus.edu

step1: Edit the crontab file with the command crontab –e

step2: The first field from the left indicates the minute's field; the second field indicates the hour's field. Since the first inject is at 2 p.m it is 14, the third field is the day of the month which is 8th, the fourth field is the month of the year which is 4(April), the fifth field is the day of the week which is Wednesday 3(Sunday-0). The path is the location where we have the text file which is being piped to the mail command which sends it with the subject work order to the email id Teama@ecs.csus.edu,similarly the second inject is set.

00 14 08 04 3 cat /ccdc_nagios/Printer_text | mail –s "work order" Teama@ecs.csus.edu

00 10 11 04 6 cat /ccdc_nagios/BLOCK_text | mail –s "work order" Teama@ecs.csus.edu

Step4: Save the file

Chapter 11

CONCLUSION

The monitoring tool Nagios is an open source tool so any user can change the source code and use it according to one's requirement. The Nagios GUI is very interactive which helps for individuals, who are not completely aware of the tool to clearly understand the status of services running on the network. Security is another major strength of Nagios which uses SSH protocol, which sends monitoring data in encrypted format across the network. The weakness of Nagios is that, to configure Nagios one needs to have a good understanding of the tool which takes lot of time

By implementing Active monitoring and passive monitoring all the services on the network are successfully monitored. Fault tolerance of Nagios is achieved through fail over monitoring which improves the reliability of the monitoring system. By implementing Distributed monitoring the load on each monitoring machine has been significantly reduced. Autoconfiguration scripts have been successfully used to configure Nagios; these scripts reduce the amount of work which needs to be done, when the number of systems to be monitored is large. Inject Automation allows Nagios to check whether the work order has been done in the given time; this again reduces the amount of work which needs to be done by white team members to check the completion of the work order.

.

Future work:

- The project can be extended by building a user interface for automated configuration using awk package in shell script.
- We can plug-in an intrusion detection to the Nagios to monitor any intrusion detection in to Nagios servers.
- The project can be extended to monitor physical intrusion by red team members in the lab allocated for the blue team.

Strengths of project:

- Reliable
- Secure
- Automatic deployment using Autoconfiguration scripts
- Easy to scale

Weakness of project:

- The project is operating system specific; it can be implemented only on Linux machines.

APPENDIX A

Acronyms

C	
CCDC	College Cyber Defense Competition.
D	
DNS	Domain Name Service.
G	
GUI	Graphical User interface.
H	
http	Hyper Text Transfer Protocol.
HTTPS	Hyper Text Transfer Protocol Secure.
N	
NRPE	Nagios remote plugin execution.
P	
PST	Pacific standard time.
R	
RDP	Remote Desktop Protocol.
S	
SMTP	Simple Mail Transfer Protocol.
SSH	Secure Shell.
T	
TCP	Transmission Control Protocol.

APPENDIX B

User Manual

NSClient++ installation steps (source: www.nagios.org)
Step1: Download NSClient++ from http://sourceforge.net/projects/nscplus

Step2: Unzip the NSClient++ files into a new C:\NSClient++ directory

Step3: Open a command prompt and change to the C:\NSClient++ directory

Step4: Register the NSClient++ system service with the following command

nsclient++ /install

Step5: Open the services manager and make sure the NSClientpp service is allowed to interact with the desktop (see the 'Log On' tab of the services manager). If it isn't already allowed to interact with the desktop, check the box to allow it to.

Step6: 7. Edit the NSC.INI file (located in the C:\NSClient++ directory) and make the following changes:

Uncomment all the modules listed in the [modules] section, except for CheckWMI.dll and RemoteConfiguration.dll

Optionally require a password for clients by changing the 'password' option in the [Settings]

Uncomment the 'allowed hosts' option in the [Settings] section. Add the IP address of the Nagios server to this line, or leave it blank to allow all hosts to connect.

Make sure the 'port' option in the [NSClient] section is uncommented and set to '12489' (the default port).

Step6: Start the NSClient++ service with the following command nsclient++ /start

NRPE installation steps :(source: www.nagios.org)

Step1: Login as a the root user

Step2: Create a new user account and give it a password

/usr/sbin/useradd nagios

passwd nagios

Step3: Create a directory for storing the downloads

mkdir ~/downloads

cd ~/downloads

Step4: Download the nagios plugins1.4.6 from nagios.org

Step5: Extract the nagios plugins source code tarball

tar xzf nagios-plugins-1.4.6.tar.gz

cd nagios-plugins-1.4.6

step6: Compile and install the plugins

./configure

make

make install

step7: Set the permissions on the plug-in directory and the plugins

chown nagios.nagios /usr/local/nagios

chown -R nagios.nagios /usr/local/nagios/libexec

step8: Install xinetd if it's already not installed with the command

yum install xinetd

step9: Download Nrpe from www.sourceforge.net and extract it in downloads folder created in step1.

tar xzf nrpe-2.8.tar.gz

cd nrpe-2.8

step10: Compile the Nrpe addon.

./configure

make all

step11: Install the Nrpe plug-in

make install-plug-in

make install-daemon

make install-daemon-config

step12: Install the NRPE daemon as a service under xinetd.

make install-xinetd

step13: Edit the /etc/xinetd.d/nrpe file and add the IP address of the monitoring server to the only_from directive.

only_from = 127.0.0.1 <nagios_ip_address>

step13: Add the following entry for the NRPE daemon to the /etc/services file.

nrpe 5666/tcp # NRPE

step14: Restart the xinetd service, with the command ‘ service xinetd restart’.

APPENDIX C

Source Code

```
#########################################################################
#####################################
#
#This script is the installation script for nagios-3.0.3 and nagios-plugins-1.4.13 prepared
by jaipal vasireddy
#email:vasireddy.jaipal@gmail.com
#date:03/25/09
#Creates log file called:nagiosinstallation.log
#
#########################################################################
#####################################

#estimating the free space under /
req=8
df -h / >size
declare -a var
while read line ;do
var=(`echo $line`)
done<size
a=`echo ${var[2]}| grep G | tr -d G`
echo the free space is $a G
if [ $a -ge $req ]
then
echo 'has sufficient space(greater than '$req' GB) for the cyber defense competition to be
conducted'
else
echo 'has insufficient space(lesser than '$req' GB)for the cyber defense competition to be
conducted'
exit
fi
#Adding user nagios
/usr/sbin/useradd -m nagios
#Prompts for nagios passwd
passwd nagios

#Adding group nagcmd
/usr/sbin/groupadd nagcmd
/usr/sbin/usermod -G nagcmd nagios
```

```
/usr/sbin/usermod -G nagcmd apache


#Prompt to enter the nagiosadmins email
echo Please enter the email of nagios admin to whom notifications need to be sent
read em


#Trying to install httpd
#checks if httpd is allready installed and it installs httpd only if its not allready
        installed

Working_Dir=`pwd`
if rpm -qa | grep httpd >nagiosinstallation.log
then
echo checked for httpd and found out that allready installed
else
echo Trying to install httpd
yum -y install httpd >>nagiosinstallation.log
out=$?
if [ $out -eq 0 ]
then
echo installed httpd successfully
else
echo installation of httpd failed refer to nagiosinstallation.log
exit
fi
fi

#Trying to install gcc
#checks if gcc is allready installed and it installs gcc only if its not allready installed
if rpm -qa | grep gcc >>nagiosinstallation.log
then
echo checked for gcc and found out that gcc allready installed
else


#Trying to install gcc
echo Trying to install gcc
yum -y install gcc >>nagiosinstallation.log
out=$?
if [ $out -eq 0 ]
```

```
then
echo installed gcc successfully
else
echo installation of gcc failed  refer nagiosinstallation.log
exit
fi
fi


#Trying to install glibc-common
##checks if glibc-common is allready installed and it installs glibc-common only  if it's
not already installed
        if rpm -qa | grep glibc-common >>nagiosinstallation.log
        then
        echo glibc-common allready installed
                else
                echo installing glibc-common
                yum -y install glibc glibc-common >>nagiosinstallation.log
                out=$?
                if [ $out -eq 0 ]
                then
                echo installed glibc-common successfully
                        else
                        echo installation of glibc-common failed
                        exit
                fi
        fi


        #Trying to install gd-devel
##checks if gd-devel is allready installed and it installs gd-devel only if it's not     already
installed
        if rpm -qa | grep gd-devel >>nagiosinstallation.log
        then
        echo gd-devel allready installed
else
echo installing gd-devel
yum install gd gd-devel >>nagiosinstallation.log
out=$?
if [ $out -eq 0 ]
then
echo installed gd-devel successfully
```

```
else
echo installation of gd-devel failed
exit
fi
fi

#Adding user nagios
/usr/sbin/useradd -m nagios
#Prompts for nagios passwd
passwd nagios

#Adding group nagcmd
#/usr/sbin/groupadd nagcmd
#/usr/sbin/usermod -G nagcmd nagios
#/usr/sbin/usermod -G nagcmd apache

#unpacking nagios-3.0.3.tar.gz
cd $Working_Dir
tar xzf nagios-3.0.3.tar.gz
cd $Working_Dir/nagios-3.0.3 >> $Working_Dir/nagiosinstallation.log
out=$?
if [ $out -eq 0 ]
then
echo nagios extraction from nagios-3.0.3.tar.gz successful
else
echo nagios extraction from nagios-3.0.3.tar.gz failed check nagiosinstallation.log
exit
fi
./configure --with-command-group=nagcmd>> $Working_Dir/nagiosinstallation.log

#compiles the nagios source code
make all >> $Working_Dir/nagiosinstallation.log
out=$?
if [ $out -eq 0 ]
then
echo "compilation of nagios source code is successful"
else
echo "compilation of nagios source code failed check nagiosinstallation.log"
exit
fi
#installing binaries
make install >> $Working_Dir/nagiosinstallation.log
```

```
out=$?
if [ $out -eq 0 ]
then
echo " installation of binaries successful"
else
echo "installation of binaries failed check nagiosinstallation.log"
exit
fi

#installing init script
make install-init >> $Working_Dir/nagiosinstallation.log
out=$?
if [ $out -eq 0 ]
then
echo "installation of init script successful"
else
echo "installation of init script failed check nagiosinstallation.log"
exit
fi

#installing sample configuration files
make install-config >> $Working_Dir/nagiosinstallation.log
out=$?
if [ $out -eq 0 ]
then
echo " installation of sample configuration files successful"
else
echo "installation of sample configuration files failed check          nagiosinstallation.log"
exit
fi

#installing command mode
make install-commandmode >> $Working_Dir/nagiosinstallation.log
out=$?
if [ $out -eq 0 ]
then
echo " installation of commandmode successfully"
else
echo "installation of commandmode failed check nagiosinstallation.log"
                        exit
                fi
```

```
#installs the nagios web configuration file in the Apache conf.d directory
make install-webconf >> $Working_Dir/nagiosinstallation.log
if [ $out -eq 0 ]
then
echo "make install-wenconf successful"
else
echo "make install-webconf failed check nagiosinstallation.log"
exit
fi
htpasswd -c /usr/local/nagios/etc/htpasswd.users nagiosadmin >>
$Working_Dir/nagiosinstallation.log
service httpd restart >> $Working_Dir/nagiosinstallation.log
cd $Working_Dir

#unpacking nagios-plugins-1.4.13.tar.gz
tar xzf nagios-plugins-1.4.13.tar.gz
cd nagios-plugins-1.4.13 >> $Working_Dir/nagiosinstallation.log
out=$?
if [ $out -eq 0 ]
then
echo "nagios-plugins extracted successfully"
else
echo "nagios-plugins do not exist check nagiosinstallation.log"
exit
fi


#compile and install the plugins
./configure --with-nagios-user=nagios --with-nagios-group=nagios >>
$Working_Dir/nagiosinstallation.log
make >> $Working_Dir/nagiosinstallation.log
out=$?
if [ $out -eq 0 ]
then
echo "make of plugins successful"
else
echo "make of plugins unsuccessful check nagiosinstallation.log"
                exit
        fi
```

```
#install plugins
make install>> $Working_Dir/nagiosinstallation.log
out=$?
if [ $out -eq 0 ]
then
echo "installation of plugins successful"
else
echo "installation of plugins failed check nagiosinstallation.log"
exit
fi#changing the default email of nagios@localhost to user specified nagiosadmin  email
sed -e 's!nagios@localhost!'$em'!'
/usr/local/nagios/etc/objects/contacts.cfg>/tmp/dupcontact.cfg
cp /tmp/dupcontact.cfg  /usr/local/nagios/etc/objects/contacts.cfg

#Add Nagios to the  list of system services and have it automatically start when
The system boots
chkconfig --add nagios >> $Working_Dir/nagiosinstallation.log
chkconfig nagios on >> $Working_Dir/nagiosinstallation.log
#Centos comes with SELinux which is security enhanced Linux and it's in
enforcing mode by default, this can result
#in "Internal Server Error "messages when we attempt to access the Nagios
CGIs, so you need to make sure that
# SELinux is in permissive mode
#check if SELinux is in enforced mode
Getenforce
#if it is enforced mode then it is changed to permissive mode
setenforce 0
echo "make sure nagios-config-text is written and then run nagios-config-files.sh"
```

Configuration script (Nagios-config-files.sh):

```
#############################################################################
#
# This script is prepared by jaipal vasireddy.
# email:vasireddy.jaipal@gmail.com
# Date:03/25/09
# This script generates a configuration file for each system specified in the nagios-config-
text file, it generates
# ccdc service, ccdc host, ccdc contact templates for any future changes in template
parameters you need to change it here
# and run the script and restart nagios
#
#
#
#
#############################################################################
####


#/bin/sh
#creating SERVICE,HOST,Contact templates
echo ' #CCDC SERVICE DEFINITION
          define service{
       name                        ccdc-service          ;
       active_checks_enabled       1                ; Active service checks are enabled
       passive_checks_enabled      1                ; Passive service checks are
enabled/accepted
parallelize_check           1               ; Active service checks should be
parallelized (disabling this can lead to major performance problems)
obsess_over_service         1               ; We should obsess over this service (if
necessary)
       check_freshness             0               ; Default is to NOT check service
'freshness'
       notifications_enabled       1                ; Service notifications are enabled
       event_handler_enabled       1                ; Service event handler is enabled
       flap_detection_enabled      1                ; Flap detection is enabled
       failure_prediction_enabled  1                ; Failure prediction is enabled
       process_perf_data           1                ; Process performance data
retain_status_information   1               ; Retain status information across program
restarts
```

```
retain_nonstatus_information 1                  ; Retain non-status information across
program restarts
                is_volatile                 0               ; The service is not volatile
      check_period                  24x7      ; The service can be checked at any time of the
day
       max_check_attempts           3     ;  Re-check the service up to 3 times in order to
determine its final (hard) state
       normal_check_interval        5                ; Check the service every 5 minutes
under normal conditions
       retry_check_interval         1                ; Re-check the service every two
minutes until a hard state can be determined
       contact_groups               admins           ; Notifications get sent out to everyone
in the 'admins' group
       notification_options         w,u,c,r          ; Send notifications about warning,
unknown, critical, and recovery events
       notification_interval        60               ; Re-notify about service problems every
hour
       notification_period          24x7             ; Notifications can be sent out at any
time
       register                     0                ; DONT REGISTER THIS DEFINITION -
ITS NOT A REAL SERVICE, JUST A TEMPLATE
       }



# CCDC host definition template

define host{
       name                         ccdc-host    ; The name of this host template
       notifications_enabled        1            ; Host notifications are enabled
       event_handler_enabled        1            ; Host event handler is enabled
       flap_detection_enabled       1            ; Flap detection is enabled
       failure_prediction_enabled   1            ; Failure prediction is enabled
       process_perf_data            1            ; Process performance data
       retain_status_information    1            ; Retain status information across program
restarts
       retain_nonstatus_information    1            ; Retain non-status information across
program restarts
       notification_period          24x7         ; Send host notifications at any time except
on november 11
       register                     0         ; DONT REGISTER THIS DEFINITION - ITS NOT
A REAL HOST, JUST A TEMPLATE!
```

```
        check_period                    24x7            ; By default, Linux hosts are checked round
the clock
        check_interval                  2               ; Actively check the host every 2 minutes
        retry_interval                  1               ; Schedule host check retries at 1 minute
intervals
                max_check_attempts              3               ; Check each Linux host 3 times
           (max)
                check_command                   check-host-alive ; Default command to check
           Linux hosts
 notification_period            workhours       ; Linux admins hate to be woken up, so we
only notify during the day
                                        ; Note that the notification_period variable is being
overridden from
                                        ; the value that is inherited from the generic-host
template!
        notification_interval           120             ; Resend notifications every 2 hours
        notification_options            d,u,r           ; Only send notifications for specific host
states
        contact_groups                  admins          ; Notifications get sent to the admins by
default
        register                        0               ; DONT REGISTER THIS DEFINITION - ITS
NOT A REAL HOST, JUST A TEMPLATE!
        }



#CCDC contact information
define contact{
 name                           ccdc-contact            ; The name of this contact template
service_notification_period     24x7    ; service notifications can be sent anytime except
on november 11th
host_notification_period        24x7    ; host notifications can be sent anytime except on
november 11th
 service_notification_options    w,u,c,r,f,s             ; send notifications for all service states,
flapping events, and scheduled downtime events
 host_notification_options       d,u,r,f,s               ; send notifications for all host states,
flapping events, and scheduled downtime events
 service_notification_commands   notify-service-by-email ; send service notifications via
email
        host_notification_commands      notify-host-by-email    ; send host notifications via
email
```

```
        register                0                       ; DONT REGISTER THIS DEFINITION -
ITS NOT A REAL CONTACT, JUST A TEMPLATE!
        }

#generic contact
define contact{
        name                            generic-contact         ; The name of this contact template
        service_notification_period     24x7                    ; service notifications can be sent
anytime
        host_notification_period        24x7                    ; host notifications can be sent
anytime
        service_notification_options    w,u,c,r,f,s             ; send notifications for all service
states, flapping events, and scheduled downtime events
host_notification_options       d,u,r,f,s               ; send notifications for all host states,
flapping events, and scheduled downtime events
        service_notification_commands   notify-service-by-email ; send service notifications
via email
        host_notification_commands      notify-host-by-email    ; send host notifications via
email
        register                0                       ; DONT REGISTER THIS DEFINITION -
ITS NOT A REAL CONTACT, JUST A TEMPLATE!
        }

# Linux host definition template - This is NOT a real host, just a template!
define host{
        name                            linux-server    ; The name of this host template
        use                             generic-host    ; This template inherits other values from the
generic-host template
        check_period                    24x7            ; By default, Linux hosts are checked round
the clock
        check_interval                  5               ; Actively check the host every 5 minutes
        retry_interval                  1               ; Schedule host check retries at 1 minute
intervals
        max_check_attempts              10              ; Check each Linux host 10 times (max)
        check_command                   check-host-alive ; Default command to check Linux
hosts
 notification_period            workhours       ; Linux admins hate to be woken up, so we
only notify during the day
                                        ; Note that the notification_period variable is being
overridden from
                                        ; the value that is inherited from the generic-host
template!
```

```
        notification_interval          120             ; Resend notifications every 2 hours
        notification_options           d,u,r           ; Only send notifications for specific host
states
        contact_groups                 admins          ; Notifications get sent to the admins by
default
        register                       0               ; DONT REGISTER THIS DEFINITION - ITS
NOT A REAL HOST, JUST A TEMPLATE!
        }
#generic host defintion
define host{
        name                           generic-host    ; The name of this host template
        notifications_enabled          1               ; Host notifications are enabled
        event_handler_enabled          1               ; Host event handler is enabled
        flap_detection_enabled         1               ; Flap detection is enabled
        failure_prediction_enabled     1               ; Failure prediction is enabled
        process_perf_data              1               ; Process performance data
        retain_status_information      1               ; Retain status information across program
restarts
        retain_nonstatus_information   1               ; Retain non-status information across
program restarts
          notification_period          24x7            ; Send host notifications at any time
except on november 11
        register                       0               ; DONT REGISTER THIS DEFINITION - ITS
NOT A REAL HOST, JUST A TEMPLATE!
        }
```

```
##############################################################################
##
##############################################################################
#######
#
# SERVICE TEMPLATES
#
##############################################################################
##
##############################################################################
#

# Generic service definition template - This is NOT a real service, just a template!

define service{
        name                            generic-service   ; The 'name' of this service template
        active_checks_enabled           1                 ; Active service checks are enabled
        passive_checks_enabled          1                 ; Passive service checks are
enabled/accepted
        parallelize_check               1                 ; Active service checks should be
parallelized (disabling this can lead to major performance problems)
        obsess_over_service             1                 ; We should obsess over this service
(if necessary)
        check_freshness                 0                 ; Default is to NOT check service
'freshness'
        notifications_enabled           1                 ; Service notifications are enabled
        event_handler_enabled           1                 ; Service event handler is enabled
        flap_detection_enabled          1                 ; Flap detection is enabled
        failure_prediction_enabled      1                 ; Failure prediction is enabled
        process_perf_data               1                 ; Process performance data
        retain_status_information       1                 ; Retain status information across
program restarts
        retain_nonstatus_information    1                 ; Retain non-status information
across program restarts
        is_volatile                     0                 ; The service is not volatile
        check_period                    24x7              ; The service can be checked at any time of
the day
        max_check_attempts              3                 ; Re-check the service up to 3 times
in order to determine its final (hard) state
        normal_check_interval           5                 ; Check the service every 5 minutes
under normal conditions
```

```
        retry_check_interval            1                       ; Re-check the service every two
minutes until a hard state can be determined
        contact_groups                  admins                  ; Notifications get sent out to
everyone in the 'admins' group
            notification_options            w,u,c,r             ; Send notifications about
warning, unknown, critical, and recovery events
        notification_interval           60                      ; Re-notify about service problems
every hour
        notification_period             24x7            ; Notifications can be sent out at any time
         register                       0                   ; DONT REGISTER THIS DEFINITION -
ITS NOT A REAL SERVICE, JUST A TEMPLATE!
        }


# Local service definition template - This is NOT a real service, just a template!

define service{
            name                            local-service           ; The name of this service
template
            use                             generic-service                 ; Inherit default
values from the generic-service definition
        max_check_attempts              3                       ; Re-check the service up to 3 times
in order to determine its final (hard) state
        normal_check_interval           5                       ; Check the service every 5 minutes
under normal conditions
        retry_check_interval            1                       ; Re-check the service every minute
until a hard state can be determined
        register                        0                   ; DONT REGISTER THIS DEFINITION -
ITS NOT A REAL SERVICE, JUST A TEMPLATE!
            }' > /usr/local/nagios/etc/objects/templates.cfg




#reads each line of config_text,for each line it creates a configuration file in
/usr/local/nagios/etc/objects and        specifies its path in the nagios main configuration
file nagios.cfg

#the below line filters lines starting with '#'in nagios-config-text and creates a file called
nagios-config-input
            cat nagios-config-text | grep -v '^#' >nagios-config-input
```

```
while read line ;do
declare -a var
var=(`echo $line | tr ',' ' '`)

 #it checks whether the host is allready declared in nagios.cfg if it is not present then it
places its path
if cat /usr/local/nagios/etc/nagios.cfg | grep ${var[1]}.cfg
then
echo "entry already present"
                                    else
                                    echo cfg_file=/usr/local/nagios/etc/objects/${var[1]}.cfg
>> /usr/local/nagios/etc/nagios.cfg
                    fi

#it creates a host definition for each host in the ccdc competition since ping is common
for  every machine i included it by default

          echo 'define host{
                    use        ccdc-host
                    host_name  '${var[1]}'
                    alias      '${var[1]}'
                    address    '${var[0]}'
                       }
                    define service{
               use                          ccdc-service         ; Name of service template to use
               host_name                         '${var[1]}'
               service_description                 PING
                         check_command                      check_ping!100.0,20%!500.0,60%
                    }' > /usr/local/nagios/etc/objects/${var[1]}.cfg

#For each service it creates a service definition

                    for (( i=2; i< ${#var[*]} ; i++ )) do
                    echo 'define service{
                        use ccdc-service
                                host_name '${var[1]}'
                                service_description  '${var[i]}'
                                check_command check_'${var[i]}'
                            }'>> /usr/local/nagios/etc/objects/${var[1]}.cfg
                    done
          #changing permission for each object file created above
        chmod 775 /usr/local/nagios/etc/objects/${var[1]}.cfg
```

```
        done <nagios-config-input
```

Command definition script (nagios-commands.sh)

```
#############################################################################
######
#
#
#This script is prepared by jaipal vasireddy
#email:vasireddy.jaipal@gmail.com
#date:03/25/09
#This script generates the command definition for each service defined in the
configuration file of each system
#in /usr/local/nagios/etc/objects/commands.cfg
#
#
#############################################################################
#####

#/bin/sh
declare -a var
#creates a empty file with the name commands.cfg
echo > /usr/local/nagios/etc/objects/commands.cfg
#filters the text file 'nagios-commands-text' begining with '#' and pipelines its output to
nagios-commands-input
cat nagios-commands-text | grep -v '^#' >nagios-commands-input
        while read line ;do
                var=(`echo $line | tr ',' ' '`)
                for (( i=2; i< ${#var[*]} ; ))do
                        #checking whether command definition is allready present
                        if cat /usr/local/nagios/etc/objects/commands.cfg | grep ${var[i]}
                        then
                        echo ${var[i]} definition allready present
                                else
                                echo '#check_'${var[i]}' using tcp
                                define command{
                                        command_name check_'${var[i]}'
                                        command_line $USER1$/check_tcp -H
$HOSTADDRESS$ -p '${var[i+1]}'
                                       }'>> /usr/local/nagios/etc/objects/commands.cfg
                        fi
```

```
                        i=`expr $i + 2`
                done
        done <nagios-commands-input
echo '# 'notify-host-by-email' command definition
define command{
        command_name    notify-host-by-email
        command_line    /usr/bin/printf "%b" "***** Nagios *****\n\nNotification Type:
$NOTIFICATIONTYPE$\nHost: $HOSTNAME$\nState: $HOSTSTATE$\nAddress:
$HOSTADDRESS$\nInfo: $HOSTOUTPUT$\n\nDate/Time: $LONGDATETIME$\n" |
/bin/mail -s "** $NOTIFICATIONTYPE$ Host Alert: $HOSTNAME$ is
$HOSTSTATE$ **" $CONTACTEMAIL$
        }

# 'notify-service-by-email' command definition
define command{
        command_name    notify-service-by-email
        command_line    /usr/bin/printf "%b" "***** Nagios *****\n\nNotification Type:
$NOTIFICATIONTYPE$\n\nService: $SERVICEDESC$\nHost:
$HOSTALIAS$\nAddress: $HOSTADDRESS$\nState:
$SERVICESTATE$\n\nDate/Time: $LONGDATETIME$\n\nAdditional
Info:\n\n$SERVICEOUTPUT$" | /bin/mail -s "** $NOTIFICATIONTYPE$ Service
Alert: $HOSTALIAS$/$SERVICEDESC$ is $SERVICESTATE$ **"
$CONTACTEMAIL$
        }


# 'check-host-alive' command definition
define command{
        command_name    check-host-alive
        command_line    $USER1$/check_ping -H $HOSTADDRESS$ -w 3000.0,80% -c
5000.0,100% -p 5
        }


# 'check_local_load' command definition
define command{
        command_name    check_local_load
        command_line    $USER1$/check_load -w $ARG1$ -c $ARG2$
        }


# 'check_local_procs' command definition
```

```
define command{
        command_name    check_local_procs
        command_line    $USER1$/check_procs -w $ARG1$ -c $ARG2$ -s $ARG3$
        }

# 'check_local_users' command definition
define command{
        command_name    check_local_users
        command_line    $USER1$/check_users -w $ARG1$ -c $ARG2$
        }


# 'check_local_swap' command definition
define command{
        command_name    check_local_swap
        command_line    $USER1$/check_swap -w $ARG1$ -c $ARG2$
        }


# 'check_ping' command definition
define command{
        command_name    check_ping
        command_line    $USER1$/check_ping -H $HOSTADDRESS$ -w $ARG1$ -c
$ARG2$ -p 5
        }


# 'check_local_disk' command definition
define command{
        command_name    check_local_disk
        command_line    $USER1$/check_disk -w $ARG1$ -c $ARG2$ -p $ARG3$
        }

# 'check_ssh' command definition
define command{
        command_name    check_ssh
        command_line    $USER1$/check_ssh $ARG1$ $HOSTADDRESS$
        }

# 'check_http' command definition
define command{
        command_name    check_http
```

```
command_line    $USER1$/check_http -I $HOSTADDRESS$ $ARG1$
}'>>/usr/local/nagios/etc/objects/commands.cfg
```

Host group definition script :(hostgroup.sh)

```
#########################################################################
#This script is developed by jaipal vasireddy
#email:vasireddy.jaipal@gmail.com
#date:
#Description:This script takes hosgroup-text as input and defines hostgroup in the
#host declared immediately after hostgrouname in hostgroup-text
#
#########################################################################

#/bin/sh
declare -a var
#removes lines beginning with '#' in file hosgroup-text and pipelines its output to
hostgroup-input

                cat hostgroup-text | grep -v '^#' >hostgroup-input
                while read line ;do
                var=(`echo $line | tr ',' ' '`)
                echo  -n 'define hostgroup{
              hostgroup_name  '${var[0]}';
              alias           '${var[0]}'; Long name of the group
              members         ' >> /usr/local/nagios/etc/objects/${var[1]}.cfg
                j=${#var[*]}
                        for((i=1;i < j-1;i++))do
                        echo -n ${var[i]}, >>
/usr/local/nagios/etc/objects/${var[1]}.cfg
                        done
                echo -n ${var[i]} >> /usr/local/nagios/etc/objects/${var[1]}.cfg
                echo -n ';' >>/usr/local/nagios/etc/objects/${var[1]}.cfg       echo '
                }'>> /usr/local/nagios/etc/objects/${var[1]}.cfg
                done < hostgroup-input
```

```
#NAGIOS-CONFIG-GEN.sh script

#reads each line of config_text,for each line it creates a configuration file in
/usr/local/nagios/etc/objects and specifies its path in the nagios main configuration file
nagios.cfg
        #the below line filters lines starting with '#'in nagios-config-text and creates a file
        called nagios-config-input
        cat nagios-config-text | grep -v '^#' >nagios-config-input
        while read line ;do
        declare -a var
        var=(`echo $line | tr ',' ' '`)

        #it checks whether the host is allready declared in nagios.cfg if it is not present
then    it      places its path
                if cat /usr/local/nagios/etc/nagios.cfg | grep ${var[1]}.cfg
                then
                echo "entry already present"
                else
        echo cfg_file=/usr/local/nagios/etc/objects/${var[1]}.cfg >>
        /usr/local/nagios/etc/nagios.cfg
                fi


        #it creates a host definition for each host in the ccdc competition since ping is
        common for  every machine i included it by default

        echo 'define host{
                 use        ccdc-host
                 host_name  '${var[1]}'
                 alias      '${var[1]}'
                 address    '${var[0]}'
                   }
                define service{
                 use       ccdc-service         ; Name of service template to use
                host_name   '${var[1]}'
                service_description             PING
                 check_command                        check_ping!100.0,20%!500.0,60%
               }' > /usr/local/nagios/etc/objects/${var[1]}.cfg
```

```
#For each service it creates a service definition

                for (( i=2; i< ${#var[*]} ; i++ )) do
                case ${var[i]} in
                dns)

                        echo "DNS name for look up on '${var[1]}' ";
                        read DNS < /dev/tty
                        echo "expected IP of '$DNS'which the '${var[1]}' DNS server
                                needs to return ";
                         read a < /dev/tty
                        echo 'define service{
                        use ccdc-service
                         host_name '${var[1]}'
                         service_description  '${var[i]}'
                         check_command check_'${var[i]}'!'$DNS'!'$a'
                                                }'>> /usr/local/nagios/etc/objects/${var[1]}.cfg

                ;;
                http)

                echo "Enter the time in days (to specify 1 day just type 1 and enter), after
                which  the  document is considered old on web server of '${var[1]}'";
                          read M < /dev/tty
                        echo "Enter the search keyword on the server side of '${var[1]}'";
                        read s < /dev/tty
                        echo 'define service{
                         use ccdc-service
                         host_name '${var[1]}'
                         service_description  '${var[i]}'
                        check_command check_'${var[i]}'!'$M'd!'$s'
                       }'>> /usr/local/nagios/etc/objects/${var[1]}.cfg

                ;;
                https)
                         echo "Enter the url of the web server on '${var[1]}'"
                        read u < /dev/tty
                        echo 'define service{
                        use ccdc-service
                        host_name '${var[1]}'
                        service_description  '${var[i]}'
                        check_command check_http!'$u'
```

```
                }'>> /usr/local/nagios/etc/objects/${var[1]}.cfg

                ;;

smtp)
        echo "Enter the name of the mail server which the remote server
                '${var[1]}' returns when Nagios sends Hello"
        read R < /dev/tty
        echo 'define service{
        use ccdc-service
        host_name '${var[1]}'
        service_description ‘${var[i]}'
        check_command check_'${var[i]}'!"HELO from white"!'$R'
                }'>> /usr/local/nagios/etc/objects/${var[1]}.cfg

  ;;
pop)
        echo "Enter the name of the pop server which the remote server
returns of '${var[1]}' returns"
        read e < /dev/tty
            echo 'define service{
            use ccdc-service
            host_name '${var[1]}'
            service_description  '${var[i]}'
            check_command check_'${var[i]}'!'$e'
                }'>> /usr/local/nagios/etc/objects/${var[1]}.cfg

  ;;


mysql)
        echo "Enter the name of the remote database on '${var[1]}' "
        read d < /dev/tty
        echo "Enter the user name of the '$d' "
        read u < /dev/tty
        echo "enter the password to access the remote database called '$d'"
         read p < /dev/tty
        echo 'define service{
        use ccdc-service
        host_name '${var[1]}'
        service_description  '${var[i]}'
        check_command check_'${var[i]}'!'$d'!'$u'!'$p'
```

```
                    }'>> /usr/local/nagios/etc/objects/${var[1]}.cfg

            ;;

rdp)

                    echo 'define service{
                    use ccdc-service
                    host_name '${var[1]}'
                    service_description  '${var[i]}'
                    check_command check_'${var[i]}'
                        }'>> /usr/local/nagios/etc/objects/${var[1]}.cfg

                        ;;
                        esac
                done

#changing permission for each object file created above
 chmod 775 /usr/local/nagios/etc/objects/${var[1]}.cfg

done <nagios-config-input
#The script check_db logs into test.student with username 'me' and password 'pass' in the local database and it then inserts a row, selects it, deletes it and#     selects it again so make sure you have the user with specified login and select,insert,delete privileges.
#service definition to monitor local Mysql database
                echo 'define service{
                use                     local-service         ; Name of service template to use
                host_name                   localhost
                service_description         DBcheck
                check_command                check_db
                notifications_enabled        0
                }' >> /usr/local/nagios/etc/objects/localhost.cfg
```

Nagios-commands-gen.sh script.

```
#########################################################################
#
#
#This script is prepared by jaipal vasireddy
#email:vasireddy.jaipal@gmail.com
#date:03/25/09
#This script generates the command definition in
/usr/local/nagios/etc/objects/commands.cfg for
#each service specified in nagios-config-text
#
#
#########################################################################

                #/bin/sh
                declare -a var
                #creates a empty file with the name commands.cfg
                echo > /usr/local/nagios/etc/objects/commands.cfg
                #filters the text file 'nagios-config-text' begining with '#' and pipelines its
                output  to nagios-commands-input
                cat nagios-config-text | grep -v '^#' >nagios-commands-input
                while read line ;do
                        var=(`echo $line | tr ',' ' '`)
                        for (( i=2; i< ${#var[*]} ; ))do
                                #checking whether command definition is allready present
                                if cat  /usr/local/nagios/etc/objects/commands.cfg | grep
                                ${var[i]}
                                then
                                echo ${var[i]} definition allready present
                                        else
                                        case ${var[i]} in
                                        dns)
                echo 'define command{
                command_name check_'${var[i]}'
                 command_line $USER1$/check_'${var[i]}' -H $ARG1$ -s
                $HOSTADDRESS$ -a $ARG2$
                                          }'>>  /usr/local/nagios/etc/objects/commands.cfg
                                        ;;



        smtp)
```

```
        echo 'define command{
         command_name check_'${var[i]}'
        command_line $USER1$/check_'${var[i]}' -H $HOSTADDRESS$
                -C $ARG1$ -R $ARG2$
                                }'>>  /usr/local/nagios/etc/objects/commands.cfg

                                 ;;
pop)
        echo 'define command{
        command_name check_'${var[i]}'
        command_line $USER1$/check_'${var[i]}' -H $HOSTADDRESS$
                -p 110 -e $ARG1$
                                }'>>  /usr/local/nagios/etc/objects/commands.cfg
                                 ;;
rdp)
        echo 'define command{
        command_name check_'${var[i]}'
         command_line $USER1$/check_tcp -H $HOSTADDRESS$ -p
                3389 -w 5 -c 8
                                }'>>  /usr/local/nagios/etc/objects/commands.cfg

                                 ;;
mysql)
        echo 'define command{
        command_name check_'${var[i]}'
        command_line $USER1$/check_tcp -H $HOSTADDRESS$ -d
                $ARG1$ -u $ARG2$ -p $ARG3$
                                }'>>  /usr/local/nagios/etc/objects/commands.cfg

esac
        fi
        i=`expr $i + 1`
        done
        done <nagios-commands-input


cp /ccdc_nagios/check_db  /ccdc_nagios/nagios-plugins-1.4.13/plugins-scripts/check_db
chmod 755 check_db
cp /ccdc_nagios/jutils.sh  /ccdc_nagios/nagios-plugins-1.4.13/plugins-scripts/jutils.sh;
echo '# 'notify-host-by-email' command definition
        define command{
              command_name    notify-host-by-email
```

```
        command_line    /usr/bin/printf "%b" "***** Nagios *****\n\nNotification Type:
        $NOTIFICATIONTYPE$\nHost: $HOSTNAME$\nState:
$HOSTSTATE$\nAddress:    $HOSTADDRESS$\nInfo:
$HOSTOUTPUT$\n\nDate/Time: $LONGDATETIME$\n" |          /bin/mail -s "**
$NOTIFICATIONTYPE$ Host Alert: $HOSTNAME$ is    $HOSTSTATE$ **"
$CONTACTEMAIL$
                }



# 'notify-service-by-email' command definition
        define command{
      command_name    notify-service-by-email
      command_line    /usr/bin/printf "%b" "***** Nagios *****\n\nNotification Type:
$NOTIFICATIONTYPE$\n\nService: $SERVICEDESC$\nHost:
$HOSTALIAS$\nAddress: $HOSTADDRESS$\nState:
$SERVICESTATE$\n\nDate/Time: $LONGDATETIME$\n\nAdditional
Info:\n\n$SERVICEOUTPUT$" | /bin/mail -s "** $NOTIFICATIONTYPE$ Service
Alert: $HOSTALIAS$/$SERVICEDESC$ is $SERVICESTATE$ **"
$CONTACTEMAIL$
      }

# 'check-host-alive' command definition
define command{
      command_name    check-host-alive
      command_line    $USER1$/check_ping -H $HOSTADDRESS$ -w 3000.0,80% -c
5000.0,100% -p 5
      }

# 'check_local_load' command definition
define command{
      command_name    check_local_load
      command_line    $USER1$/check_load -w $ARG1$ -c $ARG2$
      }

# 'check_local_procs' command definition
define command{
      command_name    check_local_procs
      command_line    $USER1$/check_procs -w $ARG1$ -c $ARG2$ -s $ARG3$
      }

# 'check_local_users' command definition
```

```
define command{
                command_name    check_local_users
        command_line    $USER1$/check_users -w $ARG1$ -c $ARG2$
        }

# 'check_local_swap' command definition
define command{
        command_name    check_local_swap
        command_line    $USER1$/check_swap -w $ARG1$ -c $ARG2$
        }

# 'check_ping' command definition
define command{
          command_name    check_ping
           command_line    $USER1$/check_ping -H $HOSTADDRESS$ -w $ARG1$ -c
$ARG2$ -p 5
        }

# 'check_local_disk' command definition
define command{
        command_name    check_local_disk
        command_line    $USER1$/check_disk -w $ARG1$ -c $ARG2$ -p $ARG3$
        }

# 'check_ssh' command definition
define command{
        command_name    check_ssh
        command_line    $USER1$/check_ssh $ARG1$ $HOSTADDRESS$
        }

#'check_http'command definition
  define command{
     command_name check_http
    command_line $USER1$/check_http -I $HOSTADDRESS$ -M $ARG1$ -s $ARG2$
                        }
#'check_https'command definition

   define command{
          command_name check_https
          command_line $USER1$/check_http -I $HOSTADDRESS$ -p 443 -S -u $ARG1$
                                    }
```

```
#check_db command definition
define command{
        command_name check_db
        command_line /ccdc_nagios/nagios-plugins-1.4.13/plugins-
    scripts/check_db
                }' >> /usr/local/nagios/etc/objects/commands.cfg
```

BIBLIOGRAPHY

[1] Omar Santos, "End –to-End Network Security: Defense-in-Depth", Cisco Press, September 2007.

[2] CCDC-network design (Copyrights Professor Richard Smith, Sacramento State University).

[3] Venkata Lakkaraju, "Database Integration And Graphical User Interface For Cyber Defense Scoring System",Project, California State University, Sacramento, FALL 2009.

[4] Competition guide for College Cyber Defense Competition, San Antonio Texas. Available: http://www.nationalccdc.org/

[5] Gregory B.White, Dwayne Williams, "Collegiate Cyber Defense Competitions", ISSA (Information Systems Security Association) Journal, October 2005.Available: https://www.issa.org/

[6] Barth Wolfgang, "System and Network Monitoring", Nagios 2nd edition, No Starch Press, October 2008.